HARDWARE BOOKS 3

6809マイコン製作実習（上）

—6809の基礎から制御用マイコンの設計まで—

PRACTICE MAKING OF 6809 MICRO COMPUTER SYSTEM

近藤元一　著

技術評論社

HARDWARE BOOKS 3

6809マイコン製作実習(上)

―6809の基礎から制御用マイコンの設計まで―

PRACTICE MAKING OF 6809 MICRO COMPUTER SYSTEM

近藤元一　著

技術評論社

はじめに

エレクトロニクスに関する技術の発展は目覚ましいものがあります．今日の最新鋭のマシンが次の日には旧型マシンに転落するかもしれないほどに技術の変遷は早く，厳しくなっています．その中でもマイクロコンピュータを中心とした電子技術の発達は“すさまじい”の一語につきるのではないでしょうか．これからのエンジニアにとって，マイクロコンピュータの理解は必須条件となることでしょう．

本書はマイクロコンピュータに興味がある人，あるいはエンジニアの卵の人にとってマイクロコンピュータの基礎を学習するための最良の参考書であることを目指してまとめたものです．

マイクロコンピュータチップには，68系の代表的な存在である6809を取り上げました．このチップを使用したワンボード・マイクロコンピュータを実際に製作し，そのコンピュータを用いて種々の機械を制御してみます．

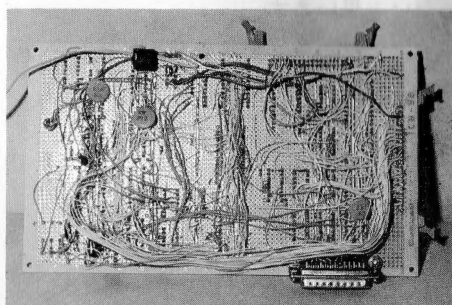
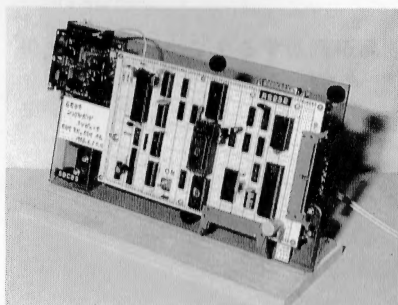
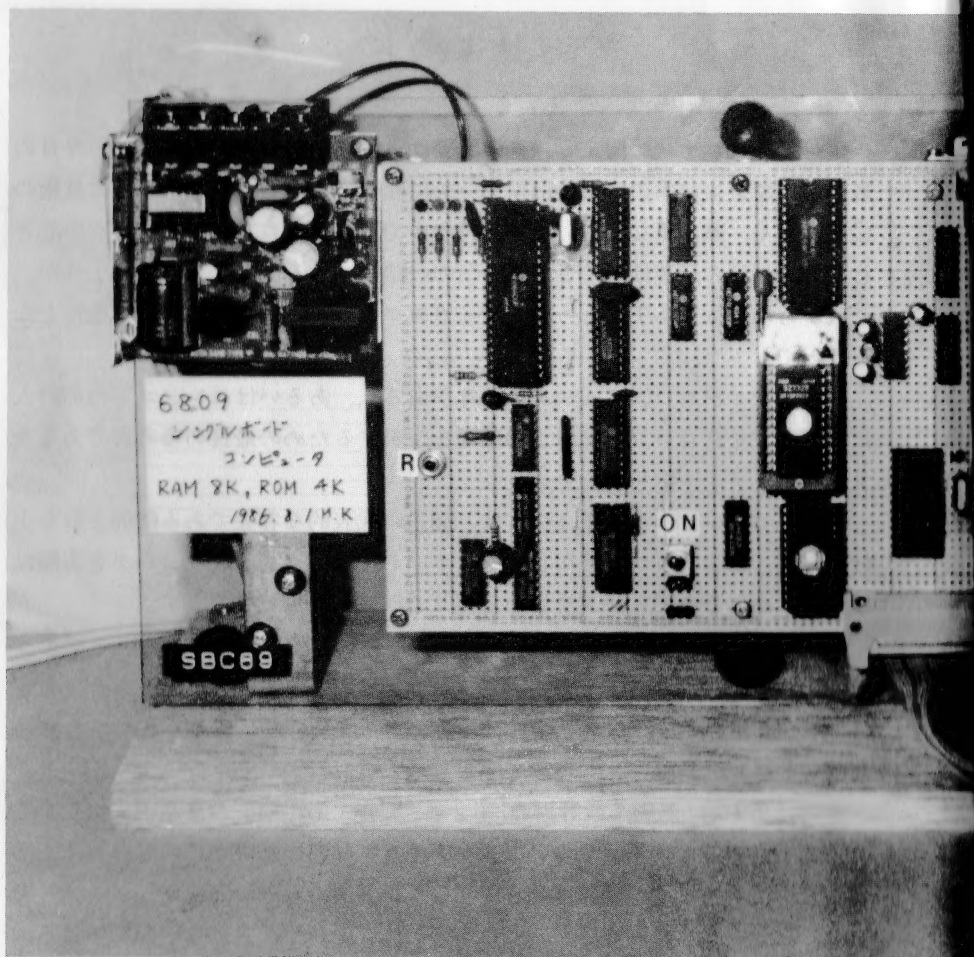
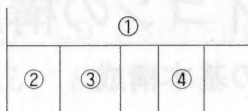
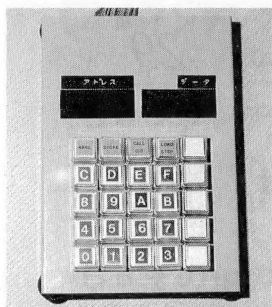
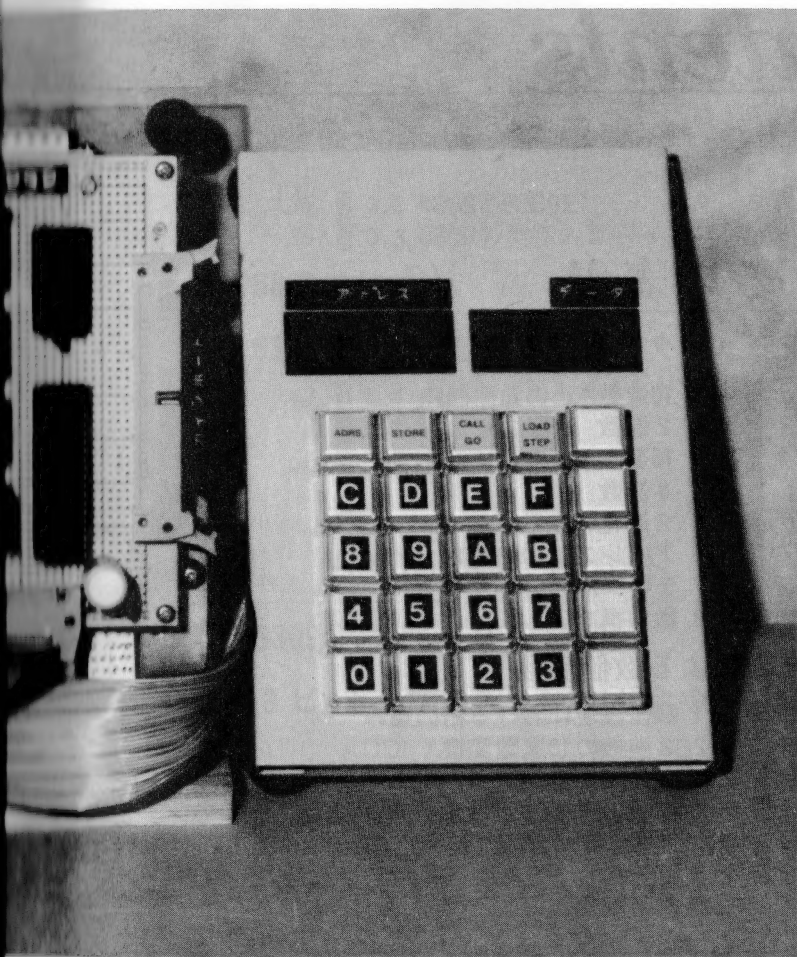
マイクロコンピュータを理解するための一番の近道は作ってみることです．100冊の本を読むよりも，1台のマイクロコンピュータを作ることの方がはるかに理解が深まります．読者も本書の内容に沿って製作してみることで，自然にソフトとハードが両方とも理解できるものと確信しています．

本書は第1巻「基礎・設計編」，第2巻「製作・応用編」の2巻構成にしました．第1巻の方でマイクロコンピュータチップ6809の基礎とその設計方法を学習します．そしてその間に部品を用意し，第2巻で実際にワンボード・マイクロコンピュータを組み立て，いろいろの機械を動かす，という手順で学習を進めたいと思います．

では，さっそく部品を発生しましょう．部品が到着するまで，じっくりと基礎から学習を行うようにしてください．

1986年12月

著者　近藤　元一

6809
シングルボード
コンピュータ
RAM 8K, ROM 4K
1986.8.1 M.K
R
ON

ワンボード・マイコン

SBC69

①SBC69外観
②基板表
③基板裏の配線
④16進キーボード

Contents

Contents

Contents

Contents

Chapter One

数の表現

1

本章では，マイクロコンピュータを学ぼうとする入門者がどうしても最低限理解しなければならない基本事項を学習します．とくに6809においては補数計算を多く使用しますので，じゅうぶんに理解してください．

これらの基本事項は大型計算機でも，マイクロコンピュータでも変わりありません．また80系や68系等の区別にも関係がない共通事項です．

よいプログラムを書くためには，本章のような基本的事項を理解しておくことが大切です．まず足下をしっかりと固めてからマイコン理解への旅へ出発することにしましょう．

1 コンピュータにおける数値表現

2進数,8進数,10進数 16進数,2進化10進数

コンピュータの世界で使われる数の表現はどうなっているでしょうか．主に使われる数の表現には，次のような５つの記数法があります．

① **２進数**（Binary Number ）

② **８進数**（Octal Number)

③ **10進数**（Decimal Number)

④ **16進数**（Hexa-decimal Number ）

⑤ **２進化10進数**（Binary Corded Decimal Number)

マイクロコンピュータの学習においては，これらの記数法が基礎になります．以下に上記にあげた記数法を詳しく説明していきます．

1･1　10進数

10進数は，日常私たちが使っている基本的な数です．０～９までの数を表す記数法です．私たちの手の指が両方合わせて10本あることから生まれた数値表現だといわれています．

1･2　２進数

２進数はコンピュータにおいてもっとも基本的な数です．０と１の２つの数で表されます．コンピュータの内部では信号は10進数は使いません．すべて２進数に変換されて情報交換が行われています．なぜ，２進数でなければいけないのでしょうか．

それはコンピュータの内部（デジタル回路）が，信号を電圧の高低のみで出力しているからです．高い場合には“H”低い場合には“L”（High, Low）と信号を出力します．１と０のみでしか表せない２進数表現とまったく同じでは

ありませんか.

それでは電圧の高低を2進数表現にあてはめてみましょう. 電圧が高い場合には"1", 低い場合には"0"に対応させます. この1と0の組み合わせで情報を作り出します. たとえばランプを8個横に並べてみます.

○○○○○○○○

そして次のようにランプを点灯して, ランプが点灯しているところを1, 消灯しているところを0と規定してみますと, 次のようになります.

○●○●○●○●

↓↓↓↓↓↓↓↓

0 1 0 1 0 1 0 1

上の図をみると, ランプの点灯状態はりっぱに1つの情報信号になっています. 次にランプを4個並べた場合に表現できる情報量がどのくらいあるか見てみましょう. 左から右のほうに並べていきます. 点灯している場合は1, 消灯している場合は0として考えます.

0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1
0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1
1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1
1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1

結果は, 0000～1111までの16個の情報量を表示することがわかりました. 2進数の表現は, 一般に数列の先頭に「%」を付けて2進数であることを表します. 例 %10010001

1・3 16進数

2進数表現はもっとも簡単な信号の組み合わせで数を表現したわけですが, 困ったことに大きい数を扱おうとすると, 非常にたくさんの2進数, つまり0と1の数を並べなくてはなりません. 私たちにとっても大変扱いにくいものです. そこで私たちにとっても, コンピュータにとっても, もっとも使いやすく, 効率的に数を表現するために考え出されたのが16進数の表現です.

16進数の表現は4桁の2進数に対し0～Fまで, つまり$16=2^4$の数を表すことができます. 先ほどの4桁の2進数に対し, 次のように数を対応させていきます. 0～9はそのまま, 10はアルファベットのAに, 11はBに, 12はCに,

13はDに，14はEに，15はFに，それぞれあてはめます．２進数，10進数，16進数の数をまとめると**表1-1**のようになります．

この表を見れば，３つの記数法がどういった対応の仕方をしているかがわかるでしょう．

10進数は９以上になると桁上がりをしますが，16進数は桁上がりをしないで１桁で表現されています．このように，10進数は桁上がりしない範囲で表現できる数が０～９（10個），16進数は０～15（16個）です．これがそれぞれ10進数，16進数の呼び名の由来となっています．

表1-1　3つの記数法

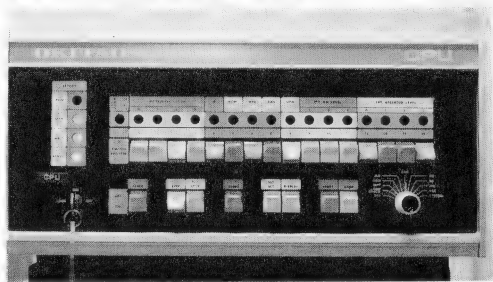
10進数	2進数	16進数
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

1・4　８進数

８進数は現在ではあまり使われていませんが，マイクロコンピュータ・チップのルーツであるｉ 8080の命令コードなどは８進数で記した方が便利です．とくにコンソール・パネルのデータセット用スイッチ（**写真1-1**）は，この記数法を使うと操作が楽にできるために相当に使われたものでした．しかし，現在では16進数の方がいろいろの面で使いやすいため，８進数の影が薄くなってしまったようです．

写真1-1　データ セット用スイッチ

16進数が２進数の列を４桁ずつに区切ったのに対し，８進数は３桁ずつ区切ったものになります．当然０～７までの数が１区切りで，８以上になると桁上がりが生じます．

なお，Octal（８進数のこと）のOをとって，123Oとか 123_8 などと表し，他の記数法と区別しています．

1・5　2進化10進数

2進化10進数は，0～9までの任意の10進数を表すのに4桁の2進数で表現する記数法です．10以上の10進数になると桁上がりが生じるので，そのときは上位に2進数4桁を追加し，10進数を表記していきます．ＢＣＤ(バイナリー・コーデッド・デシマル）とも呼ばれています．

ＢＣＤを使えば2進数を4桁ごとに区切って直読して，簡単に10進数表現に変換できます．00～99までの100とおりの数値を表現することができます．2進数のように255とおりの表現はできませんが，非常に手軽な記数法なのでよく使用されます．

たとえば68は，　0110　1000
　　　　　　　　(6)　(8)

と記されます．

表1-2　5つの記数法

10進数	2進数	16進数	8進数	BCD
0	000	0	0	0
1	001	1	1	1
2	010	2	2	2
3	011	3	3	3
4	100	4	4	4
5	101	5	5	5
6	110	6	6	6
7	111	7	7 桁上がり	7
8	桁上がり 1000	8	10	8
9	1001	9	11	9
10 桁上	1010	A	12	/ 桁上がり
11	1011	B	13	/
12	1100	C	14	/
13	桁上がり 1101	D	15	/
14	1110	E	16	/
15	1111	F 桁上がり	17 桁上がり	/
16	10000	10	20	10

※/は該当数のないことを意味する

ただ注意することは，ＢＣＤを使うときに，はっきりとＢＣＤ記法による数値であることを明示することです．そうでないと見ために２進数なのかＢＣＤなのか間違える恐れがあります．

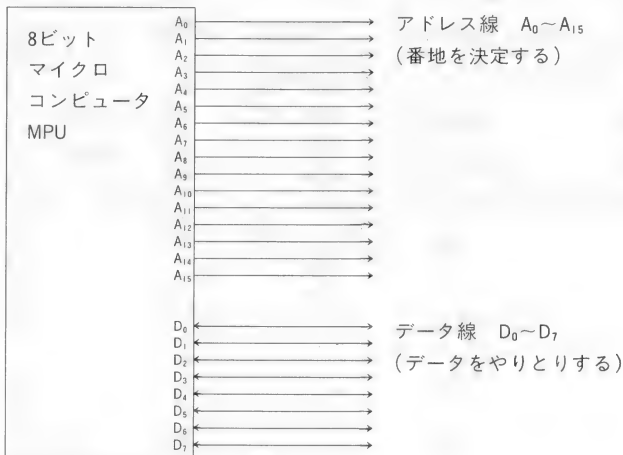
たとえば00010001という２進数があった場合，ＢＣＤ記法で10進数に直すと11という数になりますが，本来の２進数計算で10進数に直しますと17という数になります．したがって一般にＢＣＤによる場合は，はっきりＢＣＤ記法と明記することになっています．

以上，これまでの10進数，２進数，８進数，16進数，ＢＣＤの各記法による数値表現の関係は**表1-2**のようになります．

1·6　アドレス線の読みかた

さて，コンピュータ内部では信号線を使ってどのように２進数を処理しているのでしょうか．

信号線の数によって表現できる情報量は決まっています．たとえば信号線が４本であれば情報量は16，８本であれば256となります．つまり４本の信号線で表される情報量は４桁で表す２進数に対応します．この部分については前述した２進数と16進数の項目を再度読み直してもらえばはっきりとするはずです．

図1-1　アドレス線とデータ線

8ビットのマイコンには，アドレス線という信号線が16本あります（図1-1）．このアドレス線の1本1本が1か0の値をとりますので，アドレス線の16本は16桁の2進数を表現することができます．ですから，次のような表現になります．

つまり，0～1111111111111111までの数を表現できるわけです．しかし，このままでは桁数が多すぎて読み取りにくいので，これを4桁ずつに区切って，その1桁ずつを16進数表現をしてみましょう．

0000　0000　0000　0000
0　0　0　0
1111　1111　1111　1111
F　F　F　F

このように16進数で表現すると，結局0000～FFFF（$FFFFは10進数では65535）の数値が表現できます．2進数では桁数が多すぎて見やすくなかったのですが，16進数にすると桁数が少なくなり見やすくなりました．しかし16進数であるという目印がないと，10進数と区別がつかない場合があります．したがって16進数は次のような記法で明記します．

68系…………先頭に$マークを付ける　　例　$3E00

80系…………最後にHを付ける　　例　3E00H

$1000は10進数ではないのですから“セン”と発音することは誤りです．正しくは“イチ　ゼロ　ゼロ　ゼロ”と発音します．

column
コ・ラ・ム

アドレス線のとりかた

8ビット機のアドレス線は16本ありますが、番地付けがいくつできるでしょうか。

答は**表❶**のとおりです。一般によくいわれる4Kとか16Kとかいう単位はアドレスのビット値の16進数読みの4倍であることがわかりますね。この表は、一般にいっている4Kとか48Kというアドレス線の番地数の呼び方がまったくの略称であり、けっして10進数に変換したときに区切りのよい値にならないことを示しています。コンピュータでは10進数を正確な値ではなく、略称で呼ぶ方が一般的です。そのほうが人間にとって直観的に理解しやすいのです。

10進数表現ではコンピュータ内部のビットの並びと一致していないので、感覚的にデータ・バス等の変化がピンとこないのに対して、16進数であるとビット値の様子がすぐに頭に浮かびます。このようにコンピュータのハードを学習するにはビットの状況がすぐ想定できる16進数による表現が便利です。

表❶

16進表示	10進表示	情報量	略称
0000	0	1	
03FF	1023	1024	1K
07FF	2047	2048	2K
0BFF	3071	3072	3K
0FFF	4095	4096	4K
1FFF	8191	8192	8K
2FFF	12287	12288	12K
3FFF	16383	16384	16K
4FFF	20479	20480	20K
5FFF	24575	24576	24K
6FFF	28671	28672	28K
7FFF	32767	32768	32K
8FFF	36863	36864	36K
9FFF	40959	40960	40K
AFFF	45055	45056	44K
BFFF	49151	49152	48K
CFFF	53247	53248	52K
DFFF	57343	57344	56K
EFFF	61439	61440	60K
FFFF	65535	65536	64K

1・7　データ線の読みかた

前項ではアドレス線の読みかたについて考えました．では，データ線上の信号はどのようにして読むことができるのでしょうか．

８ビット機ではデータ線は８本ですので，次のような範囲で数値表現ができます．

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0
			～				
1	1	1	1	1	1	1	1

これを４桁ずつ区切ります．

D_7　D_6　D_5　D_4　D_3　D_2　D_1　D_0

①（D_7～D_4）　②（D_3～D_0）

そしてこの①，②のそれぞれを16進数で読むとどうなるでしょうか．例として次のような数値の表現の場合を考えます．

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	1	0	0	1	1

B（D_7～D_4）　3（D_3～D_0）　←　16進数による読み方

ゆえに，16進数による数値表現では＄Ｂ３であるといいます．以上のようにしてデータ線上の信号を読むことができるわけです．

1・8　数の構成

ここで，２進数の数の表し方についてまとめてみましょう．

★ビット（bit）

コンピュータで処理されるデータの最小単位です．２進数１桁で表します．

“０～１”

★ニブル（nibble）

１ビットが４個集まったデータをいいます．２進数４桁で表します．

“0000～1111”

★バイト（byte）

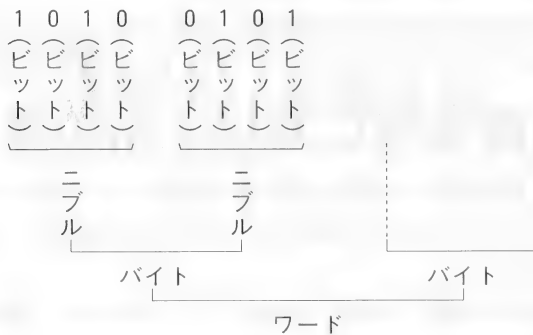
1ビットが8個集まったデータをいいます．2進数8桁で表します．

“00000000～11111111”

★ワード（word）

バイトが何個か集まったデータをいいます．8ビット機では2進数16桁で表します．

“0000000000000000～1111111111111111”

この4つの用語の関係を図に示すと次のようになります．

1と0だけの組み合わせでは判別しにくい2進数も何桁かまとめて16進数で表現すると非常にすっきりするので，積極的に上図の名称が使われています．前節で述べたＢＣＤ記数法は事務計算用のコンパイラでとくに用いられます．4ビットで1桁の10進数を表せるので10進数の1桁をニブルと呼び，この分野ではこの用語が多用されます．

またバイトは16進数2桁で表すことができます．8ビット機のデータ線は8本です．したがってデータの処理は8ビットを基本単位として行われ，この用語が使われます．マイクロコンピュータにおいて“ＬＤＡ　#＄20”という場合は2進数で“1000 0110　0010 0000”となりますが，これでは4桁に区切ってもどれくらいの数値が表現されているかは即座に判断できません．“＄86 ＄20”と16進数で表現すればわかりやすくなります．

これらの用語は，2進数を少しでも見やすいように工夫した結果，いろいろな名称になっているわけです．

2進数による数値表現

符号なし2進数と符号付き2進数

前節までの説明で2進数と16進数の関係が理解できたことでしょう．これまで例としてあげてきたのは簡単な2進数ばかりでしたが，ここではもう少し突っ込んで，2進数における負の計算方法について考えてみます．

2･*1*　数値表現における桁の重み

まず，2進数や16進数を，10進数に変換する方法について考えてみましょう．どのような記数法による数値でも各桁ごとに値を持っています．これを「数値の重み」といいます．

各桁ごとの重みを2進数と16進数の場合について考えると，次のようになります．

● 2進数　%

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

●16進数　$

3	2	1	0
16^3	16^2	16^1	16^0

各記数法で示されている数値は，この桁ごとの数値の重みに対する倍数を示しています．

たとえば8ビットの2進数01000100の場合は，

$2^6 \times 1 + 2^2 \times 1 = 64 + 4 = 68$　となります．

同様に，16進数＄0123の場合は，

$16^2 \times 1 + 16^1 \times 2 + 16^0 \times 3 = 291$　となります．

8ビット機では2進数は16桁，16進数は4桁の計算ができればじゅうぶんです．したがって先ほどの数値の重みをわかりやすい数に直すと，**図1-2** のようになります．

図1-2　2進数と16進数の桁の重み

$2^0=1$	$2^8=256$
$2^1=2$	$2^9=512$
$2^2=4$	$2^{10}=1,024$
$2^3=8$	$2^{11}=2,048$
$2^4=16$	$2^{12}=4,096$
$2^5=32$	$2^{13}=8,192$
$2^6=64$	$2^{14}=16,384$
$2^7=128$	$2^{15}=32,768$

$16^0=1$	$16^1=16$	$16^2=256$	$16^3=4,096$

2•2　2進数の10進数変換

2進数を10進数に変換する方法は，前項での「重み」を用いることによって，簡単に求められることが理解できました．それでは練習として，次の2進数を10進数に変換してみましょう．

%0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0

重みづけをすると，$2^{14}\times1+2^{11}\times1+2^8\times1+2^7\times1+2^5\times1+2^4\times1+2^2\times1+2\times1=16384+2048+256+128+32+16+4+2=18870$　となります．

もう1つの方法は一度16進数に変換してから，16進数の重みを用いて10進数に変換する手法です．

この手法を使って変換してみましょう．

重みをつけて　$16^3\times4+16^2\times9+16^1\times11+16^0\times6$
$=4096\times4+256\times9+16\times11+6$
$=16384+2304+176+6=18870$

と求めることができます．

表1-3　2の補数(8ビットの場合)

10進数		±1	±2	±126	±127	−128
2の補数	正	00000001	00000010	01111110	01111111	
	負	11111111	11111110	10000010	10000001	10000000

2•3　負の数と補数

マイナスの数はどのようにして表せばいいのでしょうか．コンピュータのハードの基本となる構成では，減算機はありません．すべて加算機で処理します．そこで加算によって減算を行う方法では**補数**(Complement) という概念を活用します．

補数は簡単に説明すれば，基数（10進数なら10，２進数なら２）から１を引いた値（２進数では２－１＝１）から各桁を引き，最後の桁に１を加えたものです．

例をあげれば，５－３→５＋(３の補数) のように処理します．この形式による処理によって減算が可能となります．したがって３の補数をハードで作るようにしておけば減算は加算で可能となり，乗算は加算の繰り返し，除算は減算の繰り返しとなり，すべて加算の処理で四則演算ができます．

補数は別名コンプリメント・ナンバー (Complement Number) といわれています．この補数には１の補数と２の補数がありますが，コンピュータの２進数の計算で使うのは，**２の補数** (2's Complement Binary Number) による場合が多いようです．８ビットの場合の２の補数を**表1-3**に示します．

正負の符号は最上位ビットが**０のときは正の数**
１のときは負の数　を表します．
ですから正の数は０～127 まで，負の数は－128 まで表現できます．

2•4　補数の重要性

なぜ，補数についてくどいほどまで説明するのでしょうか．80系ＣＰＵの本ですと，このあたりは簡単に説明されているはずなのにと思う人も多いでしょう．ところが，68系のＣＰＵ（ミニコン系も）では，実は補数が大変重要な役割をしています．

column
コ・ラ・ム

絶対アドレスと相対アドレス

絶対アドレスとは、プログラムのアドレス部に入っている値をそのままメモリの番地にするアドレス指定です。

相対アドレスとは、命令語の入っているアドレス値と目的とするデータの入っているアドレスとの差をオペランドとするアドレス指定です。

相対アドレスは絶対アドレスと比べて、次のような利点があります。

プログラムの位置移動などの変更が簡単です。また複数のプログラムを1本にまとめることが容易にできます。

68系特有の**相対アドレス**（Relative Addressing）の計算においては，ぜひとも補数によるアドレス計算が必要なのです．**絶対アドレス**（Absolute Addressing）が主になっている80系では，補数をアドレス計算に使う機会は少なく，ほとんど数値計算に使うのが大部分です．この点が68系と80系とのちょっとした違いなのです．

68系の相対アドレスにおける補数計算は2章で詳しく説明します．

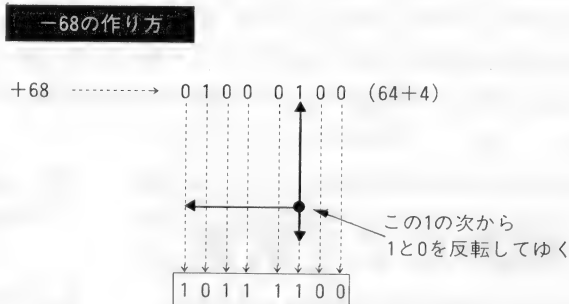
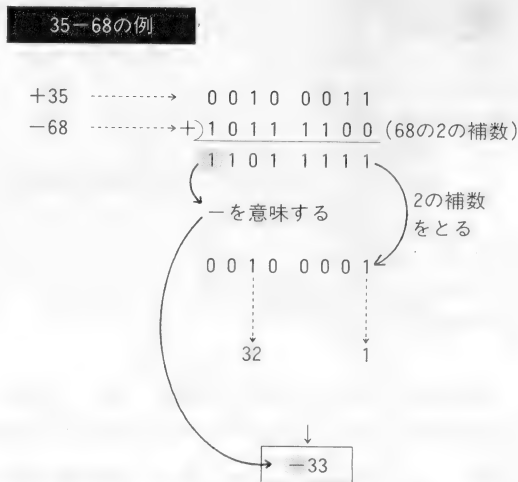
2•5　補数による計算

では，実際に2の補数を使って計算してみましょう．**図1-3**に例を示します．

この例では，35－68 を計算したわけですが，どこを見ても引算を実行しているところはありません．このように補数を使うと，加算だけで減算を行うことができます．2の補数を作るアルゴリズムは次のようになります．

『データのＬＳＢ（一番右側の最下位）からＭＳＢ側（一番左の最上位）へと数値を見ていき，最初の1が出てくるまではそのまま，1が出てきたらその1はそのままにして，次のビット値から1と0を反転していく』

このように簡単なアルゴリズムですから，論理回路を使えば簡単にＩＣで回路化できます．マイクロコンピュータの基本構成の中には，この方法がハード

図1-3　2の補数による計算例

として組み込まれています．

ちなみに6809における補数に関する命令は，COM COMA COMB の3つがあります．COMは“COMplement”の略です．

2･6　符号なし2進数と符号付き2進数

2進数には2つのタイプがあります．**単なる2進数**と**2の補数**という2つのタイプの数値です．ここでは，この2つをまとめてみましょう．
6809においては数値データを

(a)　**符号なし２進数**

(b)　**符号付き２進数**

の２つのタイプで表現します．

数値表現の方法は**図1-4**のようになります．(a)は単なる２進数です．8ビット全部を使って数値を表現しますので，０から255までの数値を表現できます．(b)は２の補数を使った２進数です．符号を考慮した２進数という意味です．一番上位のビット（ＭＳＢ）を数値の符号に使用して表現します．＋127～－128までの数値が表現できます．

この考え方は大変重要です．とくに浮動小数点演算を勉強したい人はよく理解しておく必要があります．

図1-4　6809における数値データ

(a)符号なし2進数

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	10進数
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0
∫	∫	∫	∫	∫	∫	∫	∫	
1	1	1	1	1	1	1	1	255

(b)符号付き2進数

練・習・問・題

■問1

8ビット機のビット値ですが、16進数で表すとどのようになりますか。

1)　1001　0110　0001　0110

2)　0011　0110　1100　1001

3)　0101　0101　1010　1101

〈答〉

1)9616H　2)36C9H　3)55ADH

(それぞれ$9616、$36C9、$55ADの表現にもできます)

■問2

10進数の次の数を2進数に変換しなさい。

1)　54　2)　33　3)　126　4)　232

〈答〉

1)54＝32＋16＋4＋2より　0011 0110

2)33＝32＋1より　0010 0001

3)126＝64＋32＋16＋8＋4＋2より　0111 1110

4)232＝128＋64＋32＋8より　1110 1000

■問3

次の数の2の補数を示しなさい。

1)　42　2)　28　3)　127　4)　108

〈答〉

1)42＝32＋8＋2より　1101 0110
0010 1010　→　2AH ($2A)

2)28＝16＋8＋4より　0001 1100
1110 0100　→　E4H ($EA)

3)127＝64＋32＋16＋8＋4＋2＋1より　0111 1111
1000 0001　→　81H ($81)

4)108＝64＋32＋8＋4より　0110 1100
1001 0100　→　94H ($94)

■問4

64－76＝－12の演算を補数を用い、加算だけで行う過程を示しなさい。

〈答〉

わからない人は図1-2を再度見直して、改めて計算し直してみてください。

Chapter Two

マイコンの構成

2

マイクロコンピュータはどのような構成になっていて，どんな命令で動かすかなど，多くの疑問が出てきていると思います．

本章では，MC6809というマイコンチップがどのような性質を持ち，どんな信号で動き，どのような命令で動かすかという点について，ハードとソフトの両面から説明していきます．

初心者が最初にとまどうのが用語の理解でしょう．本章あたりからいろいろの用語が出てきます．用語は機械的に理解せず，その働きや機能面の理解と結びつけて進んでいくと，マスターする速度は早まります．そのため本章では用語に関する解説をできるだけ設けるようにしました．

1 マイコンの基本構成

マイクロプロセッサ、メモリ、I/Oポート

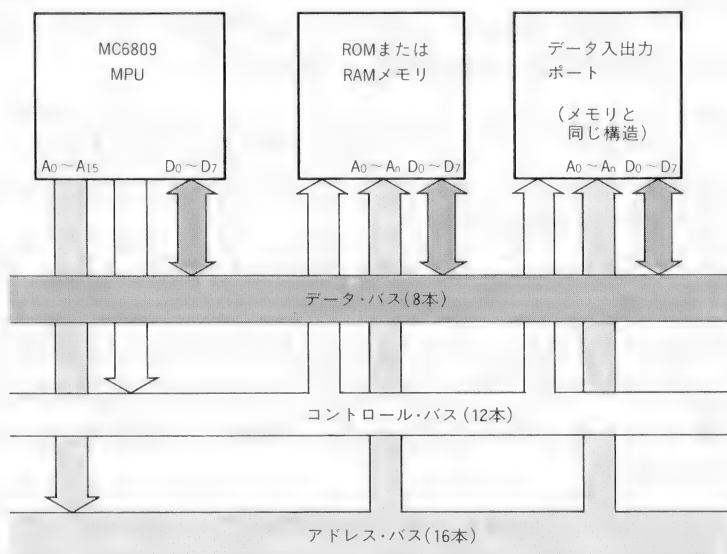
マイクロコンピュータの基本構成は**図2-1** のようになります．マイクロコンピュータは大別すると，次の３つの部分に分かれます．

①マイクロプロセッサ（ＭＰＵまたはＣＰＵ）

②メモリ

③I/O ポート（入出力ポート）

この３つの部分の基本的な役割は次のようになります．

図2-1　マイクロコンピューターの基本構成

＊マイクロプロセッサ（Micro Processing Unit ）

人間でいえば頭脳にあたるのが，マイクロプロセッサです．命令を解読したり，演算をしたり，いろいろな判断をしてコンピュータ全体の動作の流れを制御し，回路形成の中心的な役割をします．なお6809の発売元であるモトローラでは，マイクロプロセッサのことをＭＰＵと称しています．本書でも，一般的に多く呼ばれているＣＰＵ（Central Processing Unit ）の呼称を用いず，ＭＰＵと呼ぶことにします．

＊メモリ（Memory）

マイコンはハードウェアだけでなく，ソフトウェアがあって初めて動きます．メモリは，このソフトウェア（プログラム＝動作するのに必要なデータ）を記憶しておくところです．また，メモリには大きく分けてＲＡＭ（読み出しも書き込もできる）とＲＯＭ（読み出ししかできない）の２種類があり，用途によって使い分けられます．

＊I/O ポート（Input/Output port ）

ＭＰＵとメモリの構成だけでも動作しますが，マイコンの大きな目的である制御はできません．周辺の機器や回路との間でデータの授受を行う入出力用の出入口を設ける必要があります．この入出力用の出入口がI/O ポートです．ＭＰＵと周辺の機器や回路との中継基地といえます．

これらの主要部分は，バスといわれる信号通路で結ばれています．アドレス・バス上の情報をもとにして，互いにデータ・バス上で，コントロール・バスの制御によって情報を交換しています．

またバスの種類によって，信号通路の流れはさまざまです．アドレス・バスはＭＰＵから周辺に向かってのみ流れる単方向です．データ・バスはＭＰＵからデータを出力する場合と，メモリなどの出力を受けて入力する場合の２つの流れで，双方向になっています．コントロール・バスはＭＰＵからの指令を送る出力線と，ＭＰＵへの要求を受け付ける入力線の２つがありますが，個々にみれば単方向です．

これらの主要部分はアドレス・バス上の番地情報をもとにして，互いにデータバス上で情報を交換しています．

2 8ビットマシンの定義と位置づけ

8ビットマシンVS16ビットマシン

一般的にいわれている8ビットマシン，16ビットマシンという呼び方はどこからきているのでしょうか．もう一度図2-1を見てください．アドレス・バスの本数が16本，データ・バスの本数が8本になっています．原則として，このデータ・バスの本数によって何ビットマシンと呼んでいるわけです．

6809ではデータ・バスは8ビット，アドレス・バスが16ビットの容量で構成されています．では16ビットマシンと比べてはどうでしょうか．

16ビットマシンが扱える情報量は確かに飛躍的に増大しますが，必要とされる配線量もそれだけ増えます．さらにMPUのアーキテクチャも複雑化し，とても8ビットマシンの概念ではとらえきれないものがあります．

一方，8ビットマシンがいろいろな面で非常に多く活用されているのも事実です．制御面ではロボットに8ビットCPUのZ-80が，同じように自動車にも8ビットMPUの6803が使われています．

こういった事実から，8ビットマシンといえども16ビットマシンに負けない性能を備えているといえるでしょう．むしろ，コストパフォーマンス上から考えれば，制御用では8ビットマシンが最良といえるのではないでしょうか．

高度な数値計算を高速で実行させようとすれば，確かに16ビットマシン，32ビットマシンが必要とされるのは事実ですが，それほど高速さが要求されない制御面では今後も8ビットマシンの需要が続くことでしょう．その証拠に，最近（昭和60～61年）MPUチップメーカでは8ビット用MPUのμPD78312や1Mバイトの大容量メモリを扱えるHD64180R1P4,Z-80のアドレスを1MBまで拡張できるMMU，ZEN1011P等，次々に興味ある製品が生み出されています．8ビットマシンで制御の基本を学び，徐々に16ビットマシンへと移行するのがもっとも無理のない学習の姿といえます．

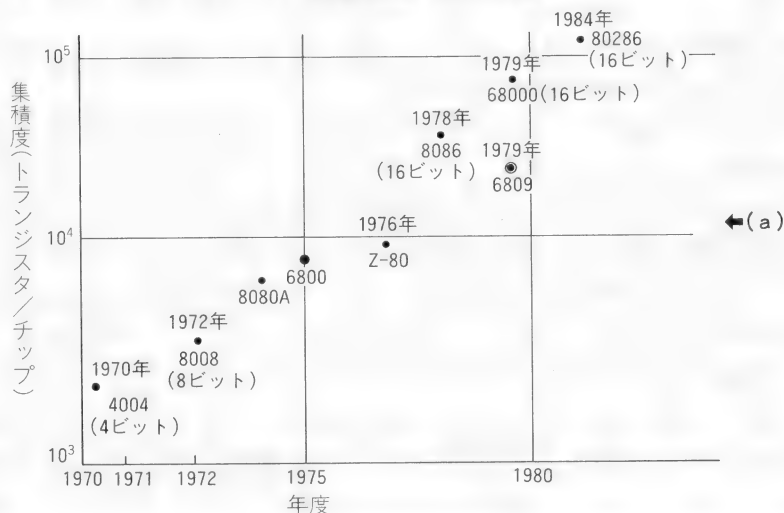
6809MPU

68系MPUと80系CPU

MC6809MPU（他社ではCPUといっている）は，アメリカのモトローラ社が生産し発売したもので，8ビットMPUとしてはもっとも高機能であるといわれています．マイクロコンピュータの発展の経緯と6809の位置づけは図2-2(a)(b) のようになります．

セカンドソースによる生産も盛んで日本では富士通，日立がセカンドソース品を供給しています．これらのチップを組み込んだ機械は私たちの目に触れないところで数多く使われています．

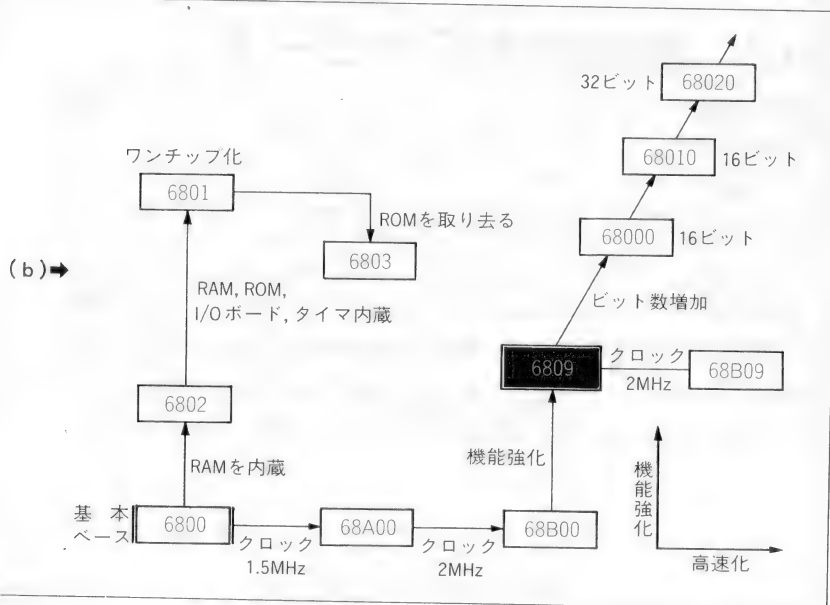
高速ラインプリンタの制御部やCRTディスプレイのコントローラに，ある

図2-2　マイクロコンピュータの発展の経緯と6809の位置づけ

いは自動車のコンピュータ制御やカーエレクトロニクスの集中管理など，多くの分野で使用されています．

パーソナルコンピュータの分野で考えるとどうしても80系ＣＰＵに圧倒されているのは事実ですが，前述したように，産業界の生産分野の開発段階では68系ＭＰＵの導入状況は80系ＣＰＵを圧倒している場合が数多く見られます．一般の人は水面上の現象だけを見て68系ＭＰＵを過小評価しているようですが，むしろ水面下では68系ＭＰＵ対80系ＣＰＵ同士の激しい戦いが続いているのです．

3・*1*　6800と6809

よく対比されることは6800と6809の関係です．モトローラ社が最初に世に出したＭＣ6800は，インテル社の i8080 とよく比較されるＭＰＵです．ＭＣ6809は，この6800と上位互換性を持たせた高性能ＭＰＵとして発表されました．

しかし6800のソフトは6809では走りますが，6809のソフトは6800では走らない場合があるのも事実です．また6800と6809ではピンはもちろん互換性はありませんし，バスのコントロール法も異なりますので，そのままでＭＰＵを差し

換えることもできません.

ただ68系の周辺ＩＣ（ファミリー）へのアクセス法は同じ方法をとっていますので，68系の周辺ファミリーはすべて使用できます．また，ＭＰＵ自体の改良も進んできたので，さらに融通性のある使用法が可能になっています.

3・2　6809と6809E

6809にはもう１つのバリエーション・バージョンがありますが，これは6809Ｅと呼ばれています．6809ＥのＥは，External（外部の）のＥをとったものです.

6809は内部に発振回路を持っており，外部に水晶振動子を接続するだけで，システム用のクロックＥとＱを作り出すことができます.

6809Ｅは，ＥとＱの基本クロックを外部の発振器から供給する構造になっています．複数のＭＰＵを駆動する形態のシステム（マルチジョブや通信方面）に効果を発揮します.

本書の製作システムでは発振器を外部に持つ6809Ｅではなく，内部に持つ6809を使用します.

3・3　6809の実行スピード

6809は実行スピードによっても，システムクロックによってそれぞれバージョンがあります．バージョンは次のようになります.

- 6809　　システムクロック　1 MH_z
- 68A09　　システムクロック　1.5 MH_z
- 68B09　　システムクロック　2.0 MH_z

高速で動かすことができればいうことはありませんが，それには速いメモリ速い周辺ＬＳＩが必要となり，金銭的な負担が増えるだけです．したがって，ＭＰＵの実行スピードを選ぶときには，無理をせず自分の構成しようとするシステムにあったバージョンを採用するようにします.

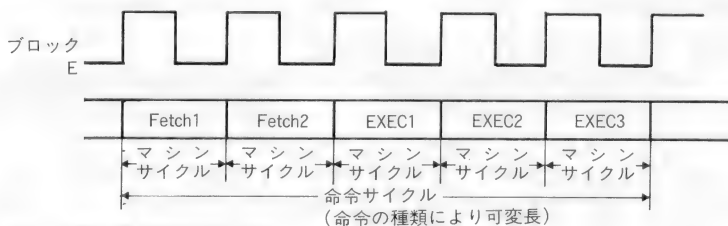
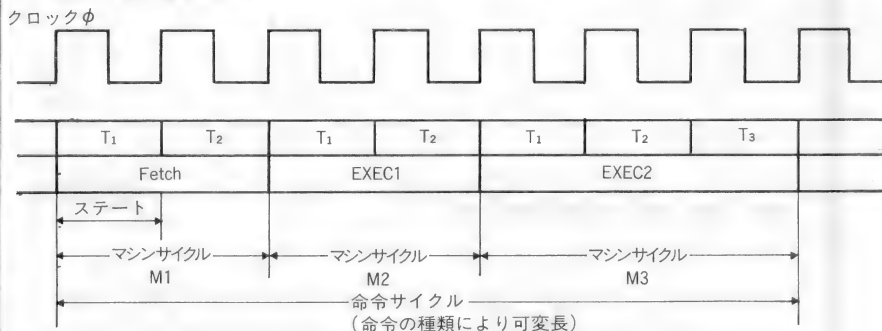
column コ・ラ・ム

80系と68系の違い

マイクロコンピュータの世界においての二大勢力はMC6809に代表される68系とZ-80やｉ8080に代表される80系であることはだれもが認めるところです．マイクロコンピュータを学ぼうとする人はこの両者の特徴をよく把握しておくことが大切です．主な相違点は次のようになります．

①バスの制御構造が違う

もっとも大きな違いがこの点です．68系はシステム全体がクロックに同期して動作します．しかし，一方の80系はクロックではなくメモリに対する読み書き信号の$\overline{RD}$，$\overline{WR}$のタイミングにより，システム全体が動作します．この関係を図❶に示します．そこで68系は同期バス，80系は非同期バスと呼ばれます．

(a) 68系の場合のタイミング

(b) 80系の場合のタイミング

Fetch：命令を取り出し解読する期間
EXEC：解読した命令を実行する期間

column
コ・ラ・ム

80系と68系の違い

図❶からわかるとおり，68系においてはマシンサイクルはシステムクロックの1周期内できちんと終了しますが，80系はクロックに同期していないのでマシンサイクルwの長さは一般に定まっていません．68系はシステム全体が1つのクロックに同期させられています．したがって，回路設計では68系のほうがタイミング設計が楽にできます．

②マシン語のアドレス値の表し方では上位バイトと下位バイトの位置が逆

たとえば＄1234番地内のデータをアキュムレータAに格納する命令は，68系では“LDA　＄1234”となります．マシン語でも素直に“＄B6　＄12　＄34”と変換されます（図❷(a)）．

しかし80系の場合は“＄3E　＄34　$12”と変換されます．アドレスの配置が68系と逆になっています(図❷(b))．68系と80系の機械語の基本形は図❸のようになります．

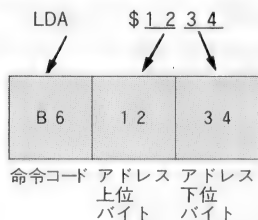
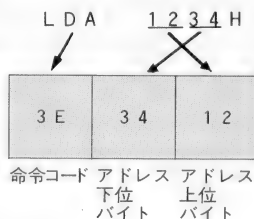
図❷　メモリ内におけるマシン語の格納状況

アドレス	マシン語
1000	B6
1001	12
1002	34
1003	⋮
⋮	

(a)

アドレス	マシン語
1000	3E
1001	34
1002	12
1003	⋮
⋮	

(b)

図❸　68系と80系の命令の基本形態

68系

80系

column
コ・ラ・ム

80系と68系の違い

③68系にはI/Oボードが存在しない

80系ＣＰＵにはメモリ域が64Ｋバイトのほかに，I/Oと呼ばれる256個の入出力制御専用のデータの出入口がありますが，68系には存在しません．その代わりにメモリ域がデータ入出力用出入口として開放されています．しかし，その分だけメモリ容量は80系よりも減少します．

④68系はメモリ中心主義であり，80系はレジスタ中心主義

68系は豊富なアドレッシングモードを使い，メモリを中心としてデータのやりとりを行いますが，80系は豊富なレジスタを使い，作業エリアにレジスタを多用したプログラムを得意とします．極端なことをいえば80系はメモリがなくてもプログラムを実行できる場合もあります．

⑤68系は相対アドレス記述中心であり，80系は絶対アドレス記述中心である

この点は68系と80系の大きな相違点です．68系のプログラムは何番地のアドレスに置いても走らせることができるリロケータブルなプログラムを容易に作成できますが，80系で同じようなプログラムを作ろうとすると大変な工夫が必要です．

以上，このほかにも68系と80系の相違点はありますが，何といっても大きな違いは8080のほうが6800よりも早く世の中にでてきたということでしょう．わずか１年くらいの差なのですが，この差が現在の使用数の違いの大きな原因となっているのです．

column
コ・ラ・ム

システムクロック

マイコンはたくさんのデバイスが連動して動いています．これらのデバイスが各自勝手なタイミングで仕事をしてしまったのでは，マイコン全体はとんでもない動きをしてしまいます．そこで回路全体が何らかの基準信号に合わせて動いていけば全体として統制がとれます．このようなシステム全体がタイミングをあわせるために出す一定間隔の基準信号をシステムクロックといいます．6809ではＥとＱの２つの信号が出されています．

6809の性能

ポジション・インデペンデント リエントラント,リカーシブ

究極の8ビットといわれる6809MPUは，どういう性能を秘めているのでしょうか．データブックに記載されている6809MPUの性能は次のようになります．

・MC6800との上位互換性を持つ．ファミリICに対しても同様
・クロックジェネレータを内蔵
・16ビット演算に対する命令を豊富に持った擬似16ビットマシンである
・強力な割り込み機能を持つ．割り込みの種類は7レベルある
・スタックはシステム・レベルとユーザ・レベルとに分かれていて，お互いに干渉しないプログラムを書くことができる
・ポジション・インデペンデント，リエントラント，リカーシブのような高機能のプログラムを書く能力を持つ
・命令は豊富であり（バリエーションは1400以上にもなる），命令体系がすっきりしている
・＋5V単一電源．TTLレベルの入出力になっていて非常に使いやすい
・内部にクロックジェネレータを装備しているので，安定したクロックを供給できる

＊6809とハンドアセンブル

6809の前身である6800が，米国DEC社のミニコン「PDP－11」の設計思想をモデルとして開発されたために，6809もミニコンの命令体系をそのまま受け継いでいます．したがって命令は非常にすっきりしていてわかりやすく，また処理スピードも80系を上回るスピードです．またアドレス方式は非常に豊富で，このアドレッシングモードとレジスタとの組み合わせによって，命令の数も1464種類にのぼります．しかし，このような数多くの命令を効率的に使える

column
コ・ラ・ム

6809ソフトウェア技法

■ポジション・インデペンデント（Position Independent）

ある番地からロードされ実行していたプログラムを，なんの変更もしないで別の番地へロードしても実行できる機能です．逆の見方をすれば，システムのアドレス空間が，そのプログラムに拘束されないともいえ，番地に対して自由であることから番地自由ともいわれます．このプログラムを容易に作成できることが68系ＭＰＵの最大の特色です．

■リエントラント（Reentrant）

割り込み処理実行中にＡというルーチンを実行しているときは別の割り込みが働き，また再びＡというルーチンが呼ばれたときには何の支障もなくそのルーチンが働く機能です．このプログラムはリエントラント・プログラムと呼ばれています．複雑にからみあった割り込み処理が必要とされる現代のマイクロプロセッサ応用技術面には必須条件といえます．

■リカーシブ・コール（Recaursive Call）

再帰呼び出し機能です．リエントラントなプログラムの飛躍した使い方で，自分が自分自身を呼んで処理を続けられることをいいます．リエントラント機能を持てば必然的にこのリカーシブル・コールも可能となります．数式評価における“（（（～）））”のような形態の処理には大きな力となります．

ようになるまでが大変です．ＭＣ6809はすっきりしていて確かにわかりやすいのですが，完全にマスターするまでが大変なＭＰＵなのです．

80系ＣＰＵでは，命令表を片手にハンドアセンブルできたものが（私も以前は6800と8080の命令を暗記していた)，6809では事実上ハンドアセンブルが不可能です．6809でハンドアセンブルしようとすれば，命令表とメモ用紙を用意し，常にアドレス，ポストバイト，レジスタ指定，オフセット値等を計算しながら進行しなければなりません．

ただ，6800に相当する命令はハンドアセンブルができますので，最初に基本部分をハンドアセンブルして，6800マシンとして立ち上げることは可能です．いずれにしても，最終的にアセンブラを採用しなければ，ＭＣ6809の機能をじゅうぶんに発揮させることは難しいでしょう．

6809の内部構成

各ピンと内部レジスタ

5・1　内部構成の概要

6809の内部構成は図2-3 のとおりです．

図2-3　MC6809の内部構成

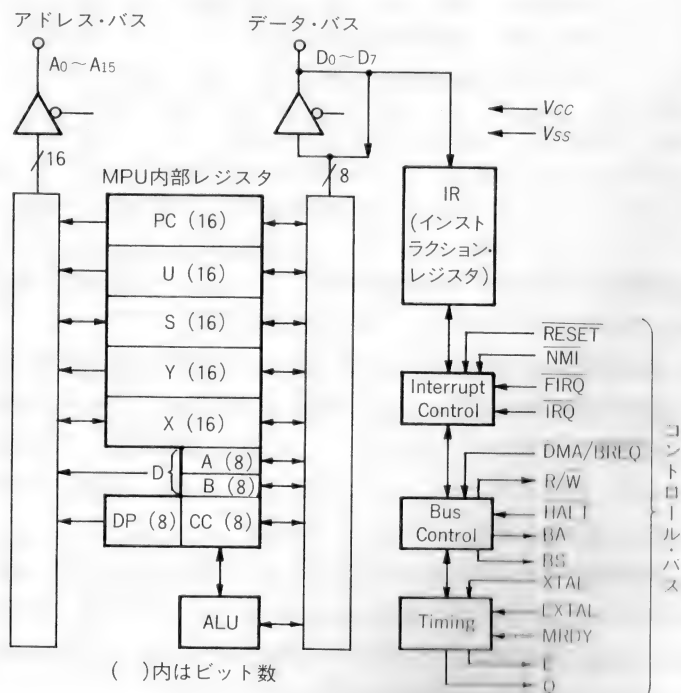
(　)内はビット数

＊ピン構成

図2-4 は6809のピン接続ですが，6809のピン構成はZ-80と同じ40ピンタイプのＬＳＩです．ピン形態は6800と似ていますが，内部構成は大きく異なり，6800との差し換えは不可能ですので注意してください．

6809は前述したように，6809と6809Ｅの２つがあります．6809は内部クロックにより駆動され，6809Ｅは外部クロックにより駆動されます．6809Ｅはマルチジョブや通信面への使用には適していますが，一般の使用ではむしろ6809が使いやすいといえます．

そこで今回の製作ではＭＰＵは6809を使用しています．

図2-4　MC6809のピン接続

ピン	信号	信号	ピン
1	V_{SS}	$\overline{\text{HALT}}$	40
2	$\overline{\text{NMI}}$	XTAL	39
3	$\overline{\text{IRQ}}$	EXTAL	38
4	$\overline{\text{FIRQ}}$	$\overline{\text{RESET}}$	37
5	BS	MRDY	36
6	BA	Q	35
7	V_{CC}	E	34
8	A_0	$\overline{\text{DMA/BREQ}}$	33
9	A_1	R/$\overline{\text{W}}$	32
10	A_2	D_0	31
11	A_3	D_1	30
12	A_4	D_2	29
13	A_5	D_3	28
14	A_6	D_4	27
15	A_7	D_5	26
16	A_8	D_6	25
17	A_9	D_7	24
18	A_{10}	A_{15}	23
19	A_{11}	A_{14}	22
20	A_{12}	A_{13}	21

＊絶対最大定格と推奨動作条件

ピンに加える電圧の絶対最大定格は**表2-1** のとおりです．絶対最大定格はＭＰＵの破壊耐力を示し，平常の使用時には**表2-2** に示す推奨動作条件の範囲内で使います．推奨動作条件の範囲外で電圧を加えて動作させたときは正常に動作しない場合があります．

＊電気的特長

一般に各ピンはＴＴＬコンパチブルとなっているので，直接に標準ＴＴＬ，ＬＳ－ＴＴＬ等と接続できます．**表2-3** の電気的特長は推奨動作条件をさらに

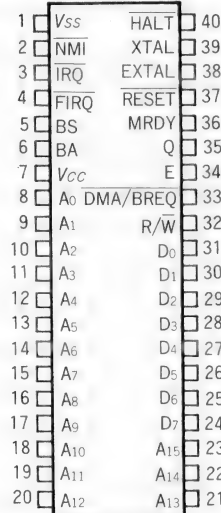



表2-1　絶対最大定格

項　　目	記　号	定　格　値	単　位
電　源　電　圧	Vcc*	−0.3～＋7.0	V
入　力　電　圧	Vin*	−0.3～＋7.0	V
動　作　温　度	Topr	−20～＋75	℃
保　存　温　度	Tstg	−55～＋150	℃

＊Vss＝0Vを基準とした値

〈注〉　絶対最大定格を越えてLSIを使用した場合，LSIの永久破壊となることがあります．また，通常動作では推奨動作条件で使用することが望ましく，この条件を越えるとLSIの信頼性に悪影響を及ぼすことがあります．

表2-2　推奨動作条件

項　　目	記　　号		min	typ	max	単　位
電　源　電　圧	Vcc*		4.75	5.0	5.25	V
入　力　電　圧	V_{IL}*		−0.3	—	0.8	V
	V_{IH}*	Logic (Ta=0～74℃)	2.0		Vcc	
		Logic (Ta=−20～0℃)	2.2		Vcc	V
		$\overline{RES}$	4.0		Vcc	
動　作　温　度	Topr		−20	25	75	°C

* Vss＝0Vを基準とした値

詳しく説明したものです.

＊内部レジスタ

内部レジスタは，５本の16ビットのレジスタ（PC，X，Y，S，U），４本の８ビットのレジスタ(AccA，AccB，CC，DP)で構成されています．なお，アキュムレータAとアキュムレータBを連結して，16ビット長のアキュムレータDとしても使用できます．

表2-3　電気的特性

●DC特性（Vcc＝5V±5％，Vss＝0V，Ta＝−20～＋75℃）

項　　目		記号	測定条件	HD6809			HD68A09			HD68B09			単位
				min	typ*	max	min	typ*	max	min	typ*	max	
入力"High"レベル電圧	$\overline{RES}$以外の入力	V_{IH}	Ta＝0～＋75℃	2.0	—	Vcc	2.0	—	Vcc	2.0	—	Vcc	V
			Ta＝−20～0℃	2.2	—	Vcc	2.2	—	Vcc	2.2	—	Vcc	
	$\overline{RES}$			4.0	—	Vcc	4.0	—	Vcc	4.0	—	Vcc	
入力"L"レベル電圧		V_{IL}		−0.3	—	0.8	−0.3	—	0.8	−0.3	—	0.8	V
入力リーク電流	EXTAL, XTAL以外の入力	Iin	Vin＝0～5.25V Vcc＝max	−2.5	—	2.5	−2.5	—	2.5	−2.5	—	2.5	μA
3ステート（OFF状態）入力電流	D_0～D_7	I_{TSI}	Vin＝0.4～2.4V Vcc＝max	−10	—	10	−10	—	10	−10	—	10	μA
	A_0～A_{15}, R/$\overline{W}$			−100	—	100	−100	—	100	−100	—	100	
出力"H"レベル電圧	D_0～D_7	V_{OH}	I_{LOAD}＝−205μA Vcc＝min	2.4	—	—	2.4	—	—	2.4	—	—	V
	A_0～A_{15} R/$\overline{W}$, Q, E		I_{LOAD}＝−145μA Vcc＝min	2.4	—	—	2.4	—	—	2.4	—	—	
	BA, BS		I_{LOAD}＝−100μA Vcc＝min	2.4	—	—	2.4	—	—	2.4	—	—	
出力"L"レベル電圧		V_{OL}	I_{LOAD}＝2mA	—	—	0.5	—	—	0.5	—	—	0.5	V
消　費　電　力		P_D		—	—	1.0	—	—	1.0	—	—	1.0	W
入　力　容　量	D_0～D_7	Cin	Vin＝0V Ta＝25℃ f＝1MHz	—	10	15	—	10	15	—	10	15	pF
	D_0～D_7 以外の入力			—	7	10	—	7	10	—	7	10	
出　力　容　量	A_0～A_{15}, R/$\overline{W}$, BA, BS	Cout		—	—	12	—	—	12	—	—	12	pF

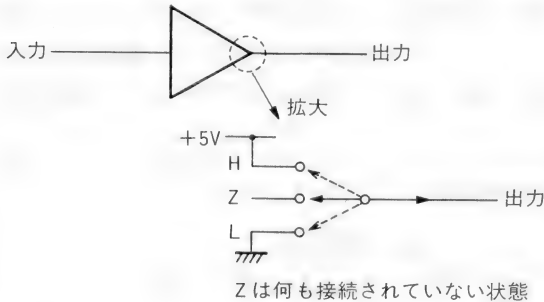
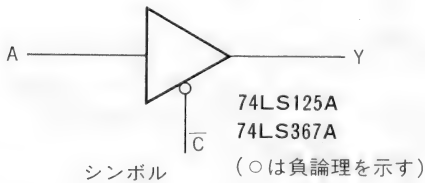
* Ta＝25℃，Vcc＝5V

column コ・ラ・ム　スリー・ステート・バッファ

デジタル回路においては一般に“H”,“L”で電圧の状態を表します．しかし，この2つの状態だけではできない状態がもう一つあります．簡単にいえば，何も接続されていない状態に近い場合です．接続されていないのですから，抵抗が非常に大きい，すなわちハイインピーダンスの状態といいます．

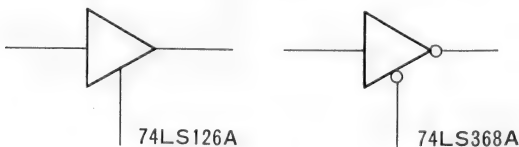
図(a)はこの3つの状態です．この3つの状態を切り換えるのが，コントロール線Cの役割です．図(b)で，Yの状態がZになっているところがハイインピーダンスの状態といい，AとYの間は断線状態になっていると考えればよいでしょう．図(c)はスリー・ステート・バッファの種類です．

(a)　出力レベルの考え方

Zは何も接続されていない状態

(b)　スリー・ステート・バッファの一例

真理値表

A	C	Y	バッファの状態
L	L	L	平常
H	L	H	平常
X	H	Z	ハイインピーダンス

(c)　いろいろのスリー・ステート・バッファ

5・2　各ピンの機能

では，各々のピンを説明していきましょう．MC6809信号の入出力方向について，ピン名称は**図2-5** のようになります．

●アドレス・バス（A_0～A_{15}）

アドレス・バスの各線はアドレスの値を出力します．これらの線は，1章で説明したように，A_0から順に，2^0，2^1，2^2，……2^{15}と名前が付けられています．

アドレス・バスでは負荷はＬＳ－ＴＴＬが4個と90ＰＦまで駆動できます．またスリー・ステート出力になっていて“0 ”，“1”の2つのレベルと，ハイインピーダンスといった何も接続されていない状態の3つの形態をとります．

●データ・バス（D_0～D_7）

データ・バスは，ＭＰＵと周辺チップとの間でデータを入力したり出力したりする通路の役目をします．このバスもアドレス・バスと同じように，トライステート出力になっています． また，負荷はＬＳ－ＴＴＬ 4個と130ＰＦまで駆動できます．

図2-5　MC6809のピンの信号の方向

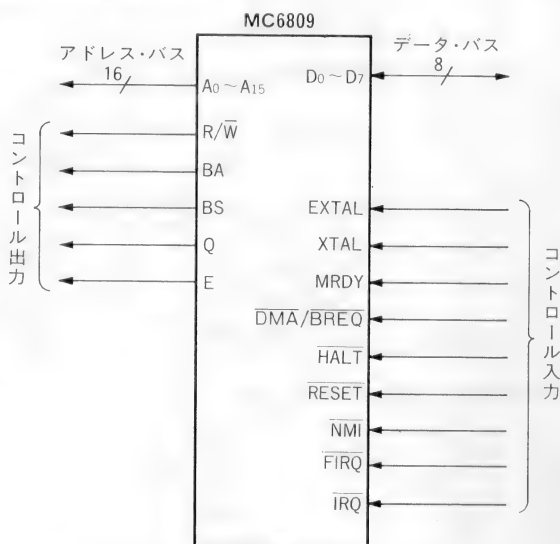
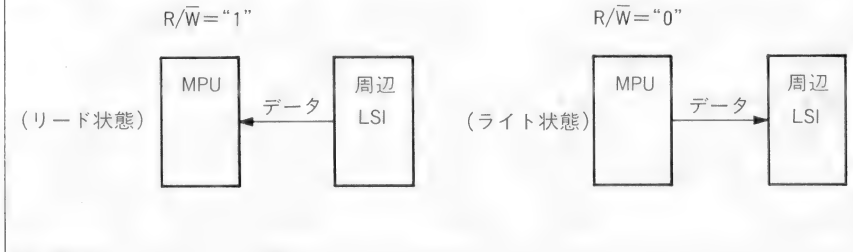



図2-6 R/W̄と入出力の関係

●リード/ライト (R/W̄)

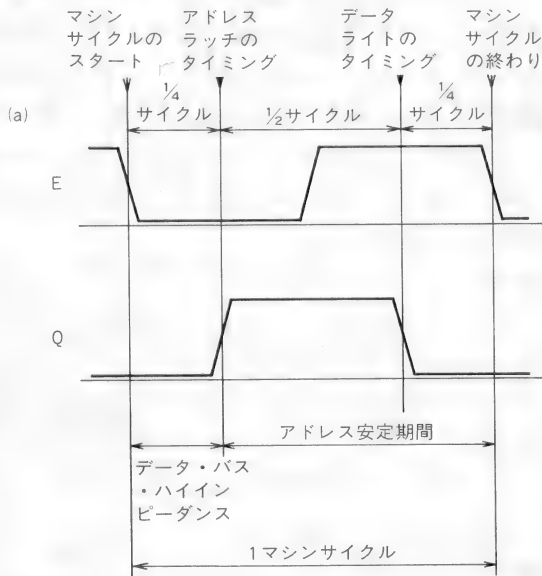
この信号は，データ・バス上のデータの移動する方向を決める信号です．READ (他のデバイスからMPUへの流れ) のときは“1”，WRITE (MPUから他のデバイスへの流れ) のときは“0”が出力されます．R/W̄のレベルと入出力の関係は図2-6 のようになります．

R/W̄の向きは，すべてMPUを中心にして見た場合を基準にしています．周辺LSIにもR/W̄のピンがありますが，周辺LSIのプログラムを考えるときも，R/W̄はあくまでMPUを主体にして考えます．

●E

E信号は6800のシステムクロック $\phi 2$ に相当します．

6809システムは，このE信号に全体のチップ・コントロールが同期して (同期バスという) 行われていますので，非常に重要です．MPUへのデータはE信号の立ち下がりで，MPUに取り込まれます (図2-7)．

図2-7 E, Q信号とバスの関係

●Q

Q信号はE信号より1/4 クロック位相が進んだタイミング用の信号です．このEとQのOR状態の期間はアドレス・バスが確定しています（**図2-7** ）．

●インタラプト・リクエスト（$\overline{\mathrm{IRQ}}$）

6809には$\overline{\mathrm{NMI}}$，$\overline{\mathrm{FIRQ}}$，$\overline{\mathrm{IRQ}}$の3つの外部割り込み端子がありますが，これらのピンを“L”にするとIRQレベルの割り込みルーチンに飛ぶことができます．これら3つの割り込み端子は，Eクロックに同期してセットする必要はありません．MPUは自分でEクロックに同期して取り込みます．

IRQ割り込みではスタックを除く全レジスタが自動的にスタックに退避します．

●ファースト・インタラプト・リクエスト（$\overline{\mathrm{FIRQ}}$）

やはり割り込み要求用の端子です．この割り込みはIRQよりも優先度が高く，またスタックにはコンデション・コード（CC）とプログラム・カウンタ（PC）の内容だけしか退避しないので処理が高速です．

●ノンマスカブル・インタラプト・リクエスト（$\overline{\mathrm{NMI}}$）

この割り込み要求は名前のとおり，ソフトによってマスクできない割り込み処理をします．割り込みの中でももっとも優先度が高い処理をします．$\overline{\mathrm{NMI}}$の“L”レベルのパルス幅は少なくとも，E信号の1サイクル以上が必要です．

●ダイレクト・メモリ・アクセス/バス・リクエクト（$\overline{\mathrm{DMA／BREQ}}$）

この信号入力はMPUのバスを外部に開放させます．端子が“L”レベルになるとそのサイクルの終わりを示し，命令は実行を中断させて，バスがMPU側から切り離されます．

●バス・アベイラブル，バス・ステータス（BA，BS）

この2つの信号により，MPUの状態を表します．BAはアドレス・バス，データ・バス，R/$\overline{\mathrm{W}}$信号がハイインピーダンス状態になっていることを表します．

BSはBAと組み合わせてMPUの状態を表します（**表2-4** ）．

表2-4　BA, BSによるMPU状態

BA	BS	MPUの状態
0	0	通常動作
0	1	インタラプト・アクノレッジ
1	0	SYNCアクノレッジ
1	1	ホールト状態

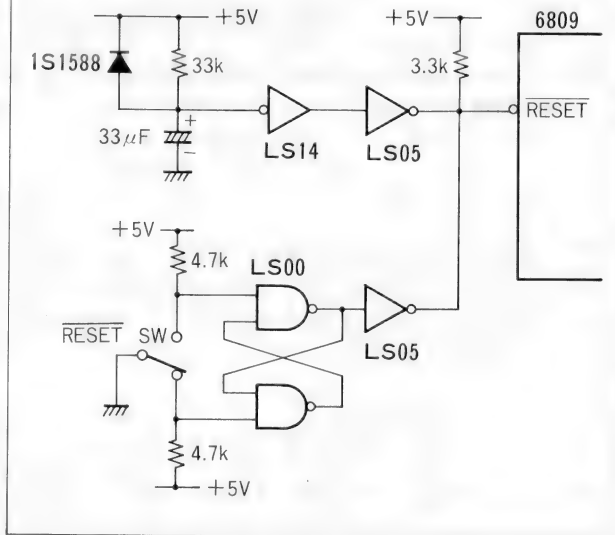
●ホールト（$\overline{\mathrm{HALT}}$）

MPUの動作を停止させます．こ

のピンを“L”レベルにすると，MPUは現在実行中の命令を終了してすぐに停止します．この間アドレス・バス，データ・バス，R/$\overline{\text{W}}$はハイインピーダンスになります．すなわち**表2-4** のようにBA＝BS＝1になります．この機能を利用して1ステップの実行ができます．

●リセット（$\overline{\text{RESET}}$）

電源を入れた直後にMPUをスタートさせる準備や，また強制的にシステムを初期化します．通常，動作中は1マシンサイクル以上“L”にすると，MPUはリセットシーケンスを実行します．注意する点は，6800はホールト（$\overline{\text{HALT}}$）期間中でもリセットが可能ですが，6809はホールト状態ではリセットできません．また電源ON後もMPUはリセットルーチンに入りますが，端子はクロックの発振が完全に安定するまで“L”にしておきます．

図2-8　リセット回路

以上の点を満足させるリセット回路は**図2-8** になります．

●メモリ・レディ（MRDY）

システムクロックE，Qの信号を引き伸ばしてMPUの動作を低速メモリに同期させます．このピンは通常“H”ですが，“L”になるとEとQが引き伸ばされ，MPUは低速メモリに対してアクセスできます．しかし最近のメモリは高速になってきているため，この端子を使うことは少なくなってきました．

●XTAL，EXTAL

水晶振動子を接続します．一般型のATカットを使用します．6809内部で，この水晶振動子の発振周波数を1/4 にして，システムクロックとしてシステム

全体に供給します.

6809と68B09における使用例は図2-9 のようになります.

column
コ・ラ・ム

マシンサイクル

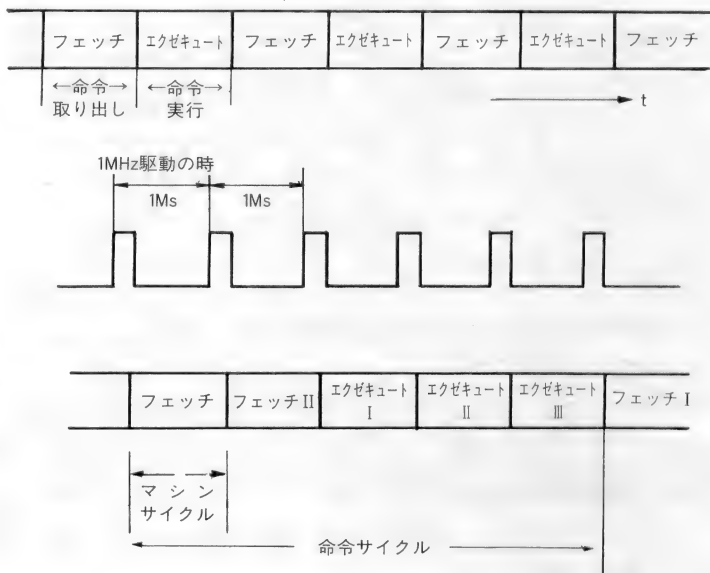
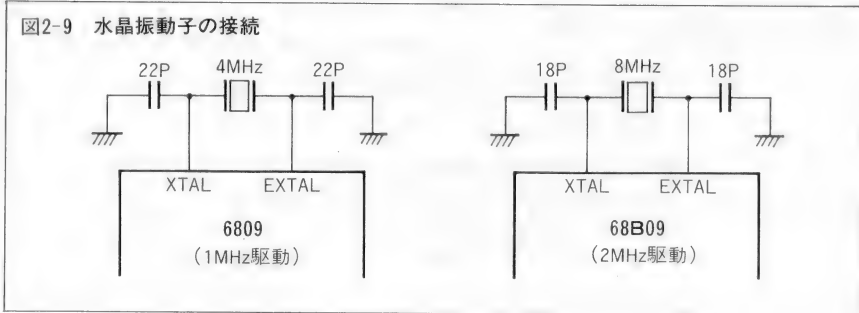
マイクロコンピュータは基本的には，まずメモリから命令を取り出してくる期間（フェッチ・サイクルという）と，その命令を実行する期間（エクゼキュート・サイクル）の繰り返しで動きます.（図❶）．この1フェッチまたは1エクゼキュートあたりの期間をマシンサイクルといいます．しかし図❶のように1つの命令は複数のエクゼキュートサイクルから構成されている場合が多いので，この命令全体を表す期間全体を命令サイクルといい，お互いに区別しています．図❶の例の場合は，この命令を実行するのに（1命令サイクル実行するのに）5マシンサイクルの時間がかかっています.

本命令を実行するのに5マシンサイクルかかる

図2-9　水晶振動子の接続

5•3　各レジスタの機能

各レジスタは図2-10のように分類されます．それでは，各レジスタの機能を説明しましょう．

●**アキュムレータ・レジスタ（AccA，AccB，AccD）**

算術演算とデータ操作に使用する汎用レジスタです．6809には8ビットのA，Bの2つのレジスタが用意されています．A，Bの各レジスタは単独で動作できるほか，Aを上位8ビット，Bを下位8ビットに設定した16ビットのアキュムレータDとしても動作できます．アキュムレータA，Bはマイクロプロセッサの中心的なレジスタです．

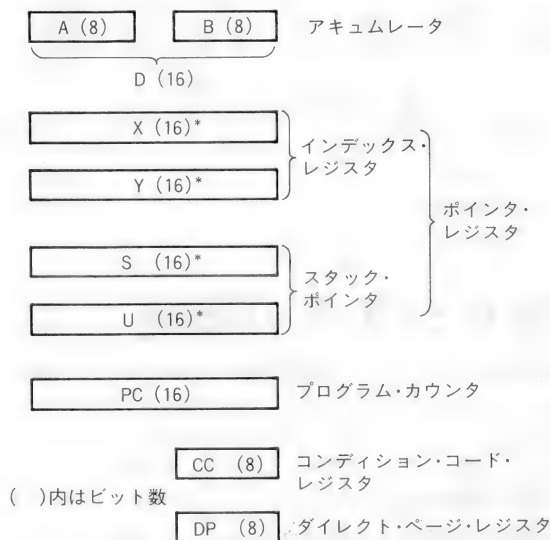
●**インデックス・レジスタ（X，Y）**

アドレス計算のときに強力な力を発揮するレジスタです．16ビットのX，Yの2つのレジスタがあります．このレジスタはMPU内にある書き込み可能な記憶装置で，計算に使うアドレスの値を自由自在に変更できます．68系のMPUの命令作成のときは，このインデックス・レジスタをうまく使えるかどうかが大きな鍵になります．Xレジスタ，Yレジスタとも同機能です．

●**スタック・ポインタ・レジスタ（U，S）**

サブルーチンへのジャンプや割り込み処理時に，各レジスタのセーブ用に使うレジスタです．16ビットのUとSの2つのレジスタがあります．一般的にUはユーザが，Sはシステムが使用します．

6809のスタック・ポインタはインデックスト・アドレッシングモード（4章P108）の能力も持っています．インデックスト・レジスタのX，Yと同じように，U，Sもアドレッシングができます．したがって6809では効率的なプログ

図2-10　MC6809のレジスタ構成

ラムを作成できます.

●プログラム・カウンタ（PC）

ＭＰＵが次に実行する命令のアドレスを示す16ビット長のレジスタです．このレジスタは，現在のコンピュータ・アーキテクチャの基本であるプログラム記憶方式を支える大黒柱です．ＭＰＵはこのレジスタを参照しながら，次の処理へと進みます.

アドレッシングモードではいろいろと活用されます.

●ダイレクト・ページ・レジスタ（ＤＰ）

ダイレクト・アドレッシングモード時（４章Ｐ106）に使用する16ビットのレジスタです．16ビット・アドレスのうち，上位８ビットの値がこのレジスタに入ります.

ＤＰに値をセットしておいてからダイレクト・アドレッシングを行うと，ＰＣの上位８ビットにＤＰの値が入り，下位８ビットにはダイレクト・アドレッシングのオペランドが入ってアドレスを確定します．6800はＤＰの中の値が擬似的に“０”の場合です.

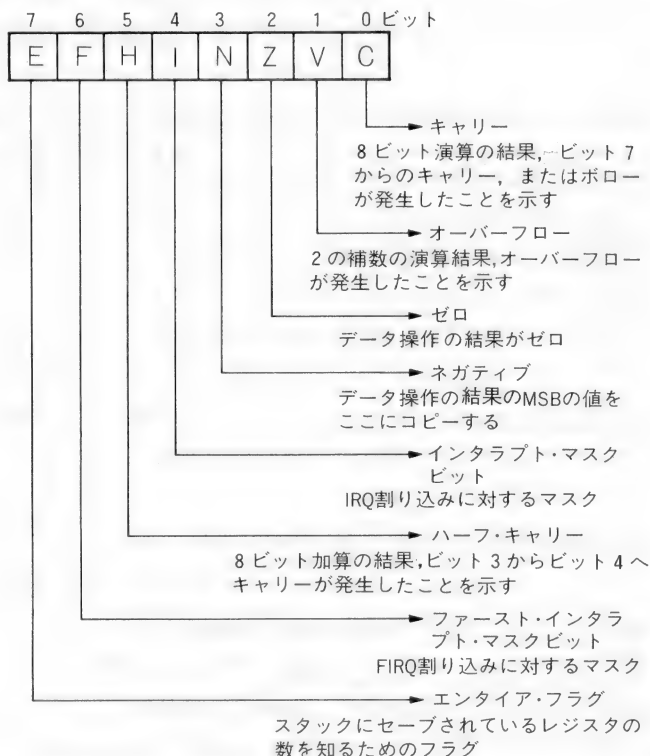
●コンデション・コード・レジスタ（CC）

演算結果の状況指示や割り込み処理に対するフラグのマスク機能を持っているレジスタです．8個のフラグで構成され，それぞれ8ビットです．レジスタの各ビットのうちC，V，Z，N，Hの各フラグは演算結果によってセットされ，I，F，Eの各フラグは割り込み処理に使われます．

コンディション・コード・レジスタの構成は図2-11のようになります．8個のフラグはそれぞれ次に述べるような働きをします．

＊キャリー（C） ビット0

8ビット演算の結果，ビット7からのキャリーまたはボローが発生したことを示します．キャリーは桁あふれが生じることで，ボローは引き算の際の上位

図2-11 コンディション・コード・レジスタの構成

のビットからの借りをいいます．

キャリーは非常に有効な機能を持っています．機能は大別すると，次の２つがあります．

（ａ）符号なし２進数において加算時のキャリー，減算時のボローを示す

図2-12のようにすることで，符号なし２進数（符号付きではない）の加減算時における桁上がり，桁下がりの発生を示すのが，Ｃフラグの役目です．キャリーとはボローの区別は，加算命令ＡＤＤのときはキャリー，減算命令ＳＵＢのときにはボローとなります．

（ｂ）数の大小判定を行う

もう１つの役目は，数の大小判定を行うことです．(ａ)で述べたように，小さい数から大きい数を引くとキャリーが１になるわけですから，比較する２つの数のうち一方をアキュムレータに入れ，他方を引き算すれば数の大小関係によってキャリーが立ったり，立たなかったりします．つまりこの状態で判定するわけです．

２つの数が等しいときは当然Ｚフラグが立ちますので，Ｚフラグも大小判定

図2-12　符号なし２進数の演算時におけるキャリーとボロー

```
1)      1 2 3          0 1 1 1 1 0 1 1
      +)  4 5        +)0 0 1 0 1 1 0 1
        1 6 8          1 0 1 0 1 0 0 0    キャリーなし

2)      1 9 6          1 1 0 0 0 1 0 0
      +)  7 2        +)0 1 0 0 1 0 0 0
        2 6 8        1 0 0 0 0 1 1 0 0    キャリー発生
                     ↖桁上がり
```

正しい答は [C] を256とみなせば
256＋12＝268となり正しい答が求まる．

```
                     [C]↘ 桁下がり発生
3)      1 2 3          0 1 1 1 1 0 1 1
      -)2 3 4        -)1 1 1 0 1 0 1 0
      - 1 1 1          1 0 0 1 0 0 0 1   補数をとる
                       0 1 1 0 1 1 1 1↙
```

桁下がりにより　256＋123－234＝145（10010001）が求まる．
この桁下がりをボロー（Borrow）という．
結局ボローが生じての計算なので，絶対値　01101111（111_{10}）は－111を意味することになる．

判断条件 / 命令	状　　態	フラグの状況
▲ BCC BHS	ACC≧M ACC≧M	C=0 C=0
*BEO	ACC=M	Z=1
*BGT	ACC>M	Z∨(N⊕V)=0
*BHI	ACC>M	C∨Z=0
▲ BCS BLO	ACC<M ACC<M	C=1
*BLE	ACC≦M	Z∨(N⊕V)=1
*BVC	オーバーフロー判定	V=0
BLS	ACC≦M	C∨Z=1
*BLT	ACC<M	N⊕V=1
BMI	ACC<M	N=1
*BNE	ACC≠M	Z=0
BPL	ACC>M	N=0
*BGE	ACC≧M	N⊕V=0
*BVS	オーバーフロー判定	V=1

* 符号付き2進数のデータの場合
▲ のグループは機能的に全く同一のグループ

表2-5
2つの符号なし2進数の大小関係と分岐命令
(1方の数ACCにもう1方の数Mに格納する)

に入れて考えると，条件付きブランチ命令（直前の演算結果によって分岐先が異なる命令）により，2つの数の大小関係の判別ができます（**表2-5**）．たとえば，ＢＣＣはCフラグが“0”のときにブランチすることを示し，ＢＣＳはCが“1”のときにブランチすることを示します．

＊オーバーフロー（V）　ビット1

演算の結果，2の補数のオーバーフローが生じたことを示します．このオーバーフローをキャリーと間違える人がいますが，キャリーとオーバーフローはまったく違います．

このフラグは符号付き2進数の演算時に有効です．演算の結果，Vフラグが立つ様子を**図2-13**により説明します．

コンピュータの内部計算では符号付き2進数で計算するケースが多いので，オーバーフローの理解は大切です．**図2-13**のように，正の数と正の数を加えたら負になるということはありえません．ですから，この状態になったらＭＰＵにそのことを教えないと，間違った計算をしてしまいます．この通知信号がオーバーフローフラグなのです．

図2-13　オーバーフローの発生

ここで注意することは，通知信号は必ず符号付き２進数の加減算命令の直後，またはＣＣのフラグをセット後に有効なことです．加減算命令以外の場合にも，このＶフラグは変化しますが，いずれも無効になりますので無視します．

このフラグをクリアーするには，

①オーバーフローの生じない符号付き２進数の加減算の実行を行う

②ＡＮＤ，ＢＩＴ，ＣＯＭ，ＥＯＲ，ＬＤＡ，ＯＲＡ，ＳＴＡ，ＴＡＢ，ＴＢＡ，ＴＳＴ，ＬＤＸ，ＬＤＳ，ＳＴＸ，ＳＴＳの各命令の実行後

の２つの場合があります．

＊ゼロ（Ｚ）　ビット２

命令を処理した後の状態が全ビット０かどうかを示します．もしＺ＝１であれば全ビットが“０”になっていることを示し，Ｚ＝０であれば“０”でないビットが１個以上あることを示しています．

＊ネガティブ（Ｎ）　ビット３

データであるＭＳＢ（もっとも最上位のビット）が１になっているとき，このフラグが立ちます．ＭＳＢの状態テストのときに使われます．符号付き２進数に関する演算後にこのフラグが“１”になったときは，結果が負の数になったことを示します．

＊インタラプト・マスク（I） ビット4

6809MPUには多くの割り込みレベルがありますが，その中の$\overline{\text{IRQ}}$入力（インタラプト・リクエスト）を受け付けるか，無視するかを決定します．

このフラグがI＝0のときにはMPUはIRQ割り込みを受け付けますが，I＝1のときには，MPUはIRQ割り込みの要求があっても無視します．ほかの割り込み，$\overline{\text{NMI}}$入力に対しては，このIフラグは関係ありません．このフラグのセット，リセットはプログラムで行います．

＊ハーフ・キャリー（H） ビット5

ハーフ・キャリーの名前のとおり，ビット3からビット4への桁上がりを表すフラグです．8ビットのMPUにおけるアキュムレータは，ビット0～ビット7までを2つに分割して（ビット0～ビット3とビット4～ビット7）それぞれをハーフといいます．ADD，ABA，ADC命令を実行すると，その直後にビット3からビット4にキャリーが立ったことを知らせます．

＊ファーストIRQマスク（F） ビット6

Iフラグと同様に，6809MPUにおける$\overline{\text{FIRQ}}$入力に対するマスクフラグです．このビットが“1”のときFIRQ割り込みを無視し，“0”のときイネーブルとなり実行します．リセットのときは“1”になり，NMI，SWI，FIRQが受け付けられるとCCがスタックにセーブされ，次に“1”になります．なおIRQ，SWI2，SWI3はFビットとは関係ありません．このフラグもプログラムでセット，リセットを行います．

＊エンタイア（E） ビット7

MPUは割り込み処理終了後元のルーチンへ戻りますが，そのときにスタックにセーブされた各レジスタを元に戻しておきます．この場合，IRQとFIRQではセーブされているレジスタの数が異なります．この数の違いを教えるのが，エンタイア・フラグの役目です．RTI命令のみが参照できます．

このフラグが“1”のときはSP（スタック・ポインタ）を除く，全レジスタ(Entire Register)がセーブされていることを示しRTI命令が実行されます．そして元のルーチンに戻るときMPUはEフラグを参照して，CCとSP以外の全レジスタを元に戻します．また，このフラグが“0”のときはPCだけが戻されます．このように，Eフラグはハードウェア（MPU）が自動的にセット，リセットを行います．

Chapter Three

6809命令セット

3

これまで述べてきたようにMC6809はひじょうに多くの命令を持っています．これらの命令を適切に使用することで，効率のよいプログラムが作ることができます．そのためには一つ一つの命令の理解が大切です．

本章では，MC6809の数多くの命令を，まず機能面を中心に分類して，命令の働きを説明していきます．命令語にはすべて英語のフルスペルをつけておきました．できるだけフルスペルで覚えるのが命令を身近なものにできるコツといえましょう．

最初はロード命令とストア命令を重点的に学習するのが無理がないでしょう．どのようなコンピュータにおいても，この2つの命令が基本になっていますので，命令の意味をまず，じゅうぶんに理解してみましょう．

1 6809命令セットの分類法

5つのグループ命令

6809の命令はニーモニック・コードで数えると59になります（**表3-1** ）．しかし，これはあくまで命令の基本形態であり，このコードが後述のアドレッシングモードの違いによりさらに分類されていき，最後には1464種類もの数になります．このような多くの命令をすべて覚えるのは不可能ですが，機能によって分類していくと，かなりすっきりしたものになります．

本章では，理解しやすいように演算対象のレジスタを中心として，アルファベット順に5つのグループに分類してみました（巻末命令参照）．

①8ビット・アキュムレータ／メモリ命令
②16ビット・アキュムレータ／メモリ命令
③インデックス・レジスタ／スタック・ポインタ命令
④ブランチ命令
⑤その他の命令

表3-1　59種類の6809命令

ニーモニック・コード	動作内容	ニーモニック・コード	動作内容
ABX	Add Accumulator B into Index Register X	DAA	Decimal Addition Adjust
ADC	Add with Carry into Register	DEC	Decrement
ADD	Add Memory into Register	EOR	Exclusive OR
AND	Logical AND Memory into Register	EXG	Exchange Registers
ASL	Arithmetic Shift Left	INC	Increment
ASR	Arithmetic Shift Right	JMP	Jump
BCC	Branch on Carry Clear	JSR	Jump to Subroutine
BCS	Branch on Carry Set	LD	Load Register from Memory
BEQ	Branch on Equal	LEA	Load Effective Address
BGE	Branch on Greater Than or Equal Zero	LSL	Logical Shift Left
BGT	Branch on Greater	LSR	Logical Shift Right
BHI	Branch if Higher	MUL	Multiply
BHS	Branch if Higher or Same	NEG	Negate
BIT	Bit Test	NOP	No Operation
BLE	Branch if Less than or Equal to Zero	OR	Inclusive OR Memory into Register
BLO	Branch on Lower	PSH	Push Registers
BLS	Branch on Lower or Same	PUL	Pull Registers
BLT	Branch on Less than Zero	ROL	Rotate Left
BMI	Branch on Minus	ROR	Rotate Right
BNE	Branch Not Equal	RTI	Return from Interrupt
BPL	Branch on Plus	RTS	Return from Subroutine
BRA	Branch Always	SBC	Subtract with Borrow
BRN	Branch Never	SEX	Sign Extend
BSR	Branch to Subroutine	ST	Stor Register into Memory
BVC	Branch on Overflow Clear	SUB	Subtract Memory from Register
BVS	Branch on Overflow Set	SWI	Software Interrupt
CLR	Clear	SYNC	Synchronize to External Event
CMP	Compare Memory from a Register	TFR	Transfer Register to Register
COM	Complement	TST	Test
CWAI	Clear CC bits ane Wait for Interrupt		

2 8ビット・レジスタ/メモリ命令

基本的な命令

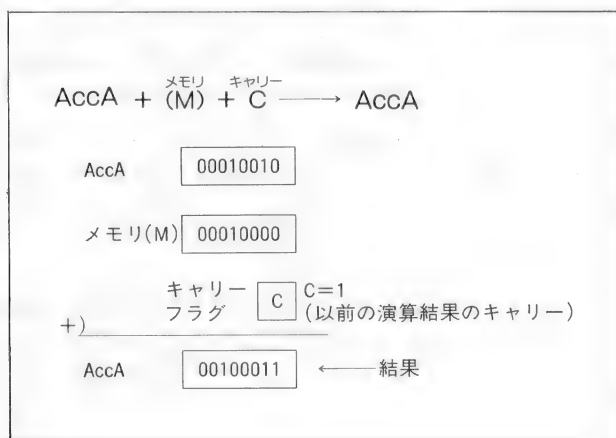
このグループにまとめた命令群は演算の対象が8ビットのレジスタであるアキュムレータA（AccA），B（AccB）と1バイト（8ビット）のメモリMに限定されています．

基本的な命令ばかりですので，わかりやすい素直な形態になっています．ここで説明する命令は27種類です．

ADC

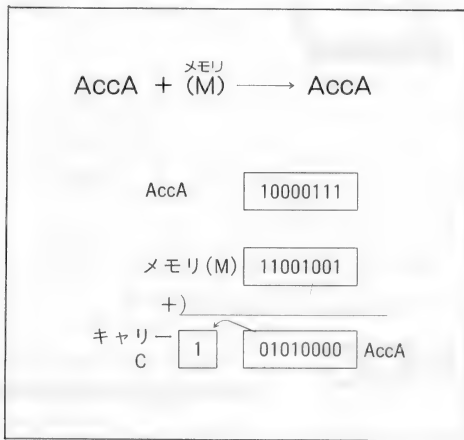
(ADd memory to accumulator with Carry)

キャリー（Cフラグ）とメモリの内容をアキュムレータに加算します．メモリは当然1バイトです．結果はアキュムレータ中に残ります（アキュムレータの内容のみが変更される）．

ソース形式例　　ADCA　；　ADCB

ADD

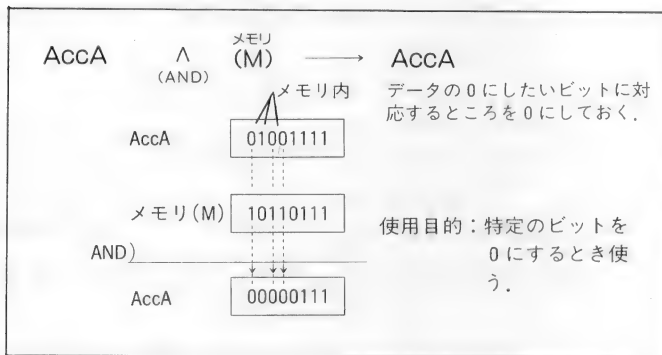
(ADD memory to accumulator)

メモリの内容（1バイト）をアキュムレータに加算します．結果はアキューームレータ中に残ります．

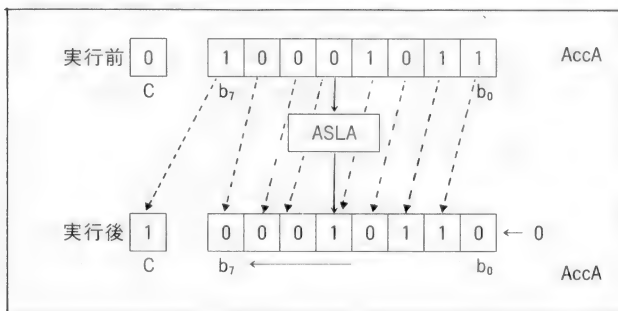
ソース形式例　　ADDA　；　ADDB

AND

(AND memory with accumulator)

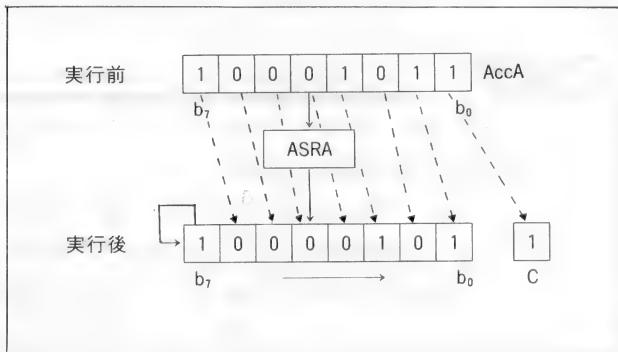
アキュムレータとメモリの内容とで論理積を行います．結果はアキュムレータ中に残ります．メモリ内データの特定のビットを0にするときに用います．

ソース形式例　　ANDA　；　ANDB

ASL (Arithmetec Shift Left)

アキュムレータまたはメモリの内容を全ビット１つ左へシフト（ずらすこと）します．後述のＲＯＬ命令とは違って，ビット０には０が入り，またビット７はＣフラグへシフトされます．

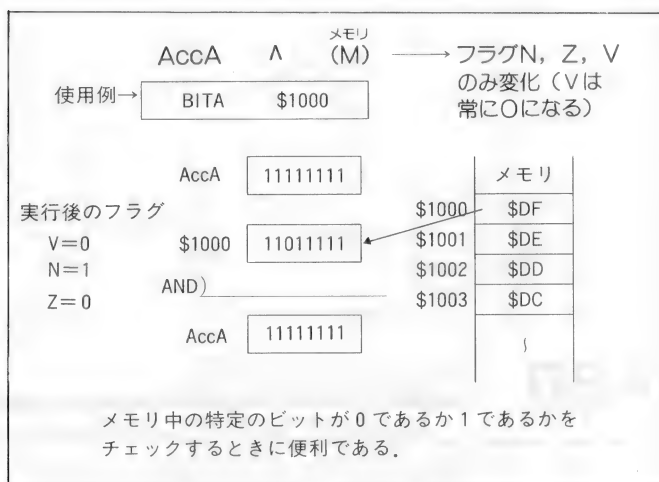
ソース形式例　　ＡＳＬ　；　ＡＳＬＡ　；　ＡＳＬＢ

ASR (Arithmetic Shift Right)

アキュムレータまたはメモリの内容を全ビット１つ右へシフトします．後述のＲＯＲ命令とは違って，ビット０はＣフラグへシフトされますが，ビット７の値はそのまま保存されます．

ソース形式例　　ＡＳＲ　；　ＡＳＲＡ　；　ＡＳＲＢ

BIT

(BIT test)

アキュムレータとメモリの内容との論理積をとり，その結果によりCC（コンディション・コード）を更新します．アキュムレータの中身は変化しません．結果を格納しないAND命令といえます．

ソース形式例　BITA　；　BITB

CLR

(CLeaR)

アキュムレータとメモリの内容を0にセットします．

ソース形式例　CLR　；　CLRA　；　CLRB

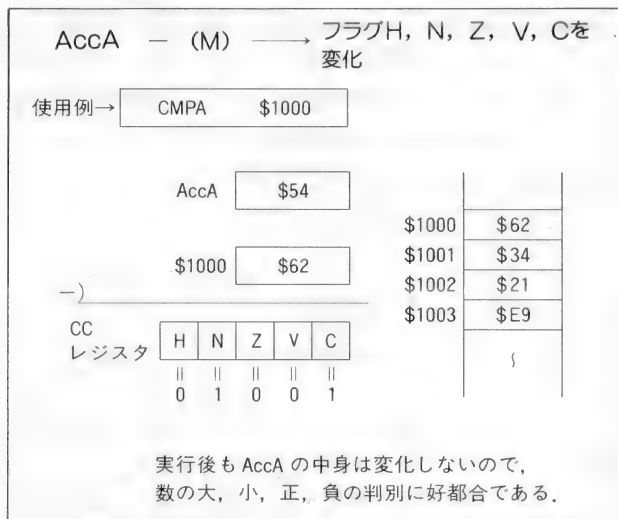
CMP

(CoMPare memory from accumulator)

レジスタの内容からメモリの内容を差し引きして，CCのみを変化させます．

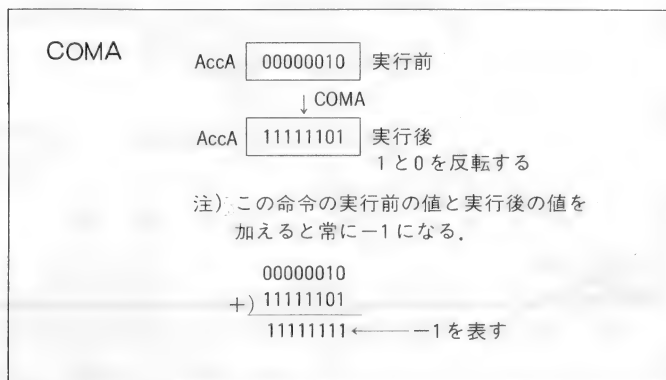
同じような命令にSUB命令がありますが，SUB命令は演算結果がレジスタに残り，CMP命令の場合はレジスタに演算結果は残りません．これはCMP命令が，レジスタとメモリ内容との比較に目的がおかれているからです．その結果

CCのN，Z，Q，C，Hの各フラグが変化します．主に比較する両者の大小関係（等号も含む）の判定に多用されます．

ソース形式例　　CMPA　；　CMPB

COM (COMplement)

アキュムレータとメモリの内容について1と0の反転を行います．CCのキャリーフラグは必ず“1”になっています．

ソース形式例　　COM　；　COMA　；　COMB

DAA

(Decimal Addition Adjust)

ＢＣＤ（２進化10進数）で数値を処理しているとき，下位４ビット同士の演算で桁上げが生じますが（ハーフキャリーが立った），このハーフキャリー（Ｈフラグ）を上位４ビットの値に加えることで，10進補正を自動的に行います．

ソース形式例　　ＤＡＡ

DEC

(DECrement)

指定されたレジスタまたはメモリの内容を－１します．キャリーフラグは全然変化しないので，ＣＣによる条件判定の

ときには注意してください．

ソース形式例　DEC　；　DECA　；　DECB

EOR

(Exclusive OR)

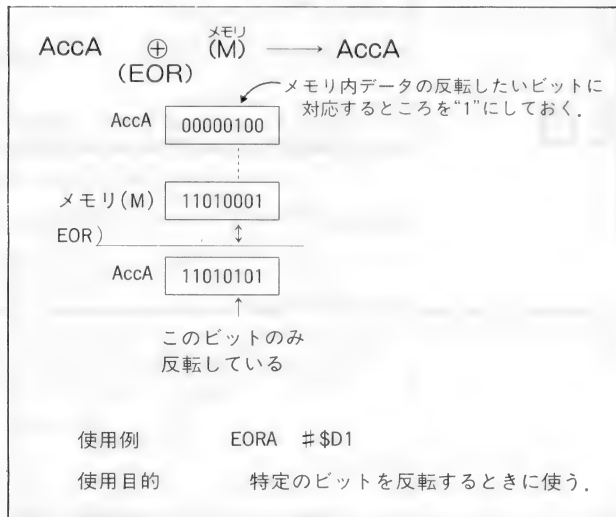
アキュムレータとメモリの内容との排他的論理和を求めます．結果はアキュムレータに残ります．メモリ内の特定のビットを反転するときに用います．両方の値が異なるときに"1"になり，等しいときに"0"になります．排他的論理和を記号⊕で表すと，次のような関係式になります．

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

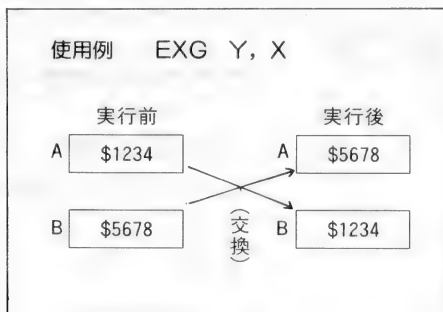
$1 \oplus 1 = 0$

ソース形式例　EORA　；　EORB

EXG

(EXchanGe)

2つの指定した8ビット・レジスタの内容を交換します．レジスタとしてはA，B，CC，DPが指定できます．なお

交換するレジスタ同士はビット長が同じことが必要です.

ソース形式例　　EXG

INC

(INClement)

DEC命令と逆の動作をします．指定されたレジスタまたはメモリの内容を＋１します．キャリーフラグが変化しない点も同じです．

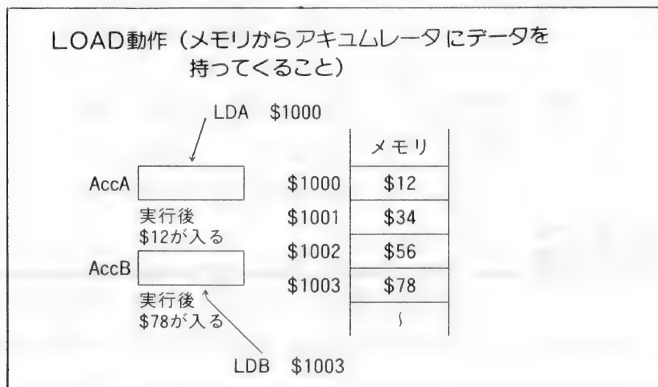
ソース形式例　　INC　；　INCA　；　INCB

LD

(LoaD accumulator from memory)

マイクロコンピュータの命令中において，もっとも基本的な命令の一つです．メモリの内容を，指定された８ビット・レジスタにコピーします．ST命令とは逆の動作をします．

ソース形式例　　LDA　；　LDB

LSL

(Logical Shift Left)

前述したASL命令とまったく同じ動作をします．ですからマシン語コードも同一です．

ソース形式例　　LSL　；　LSLA　；　LSLB

LSR

(Logical Shift Right)

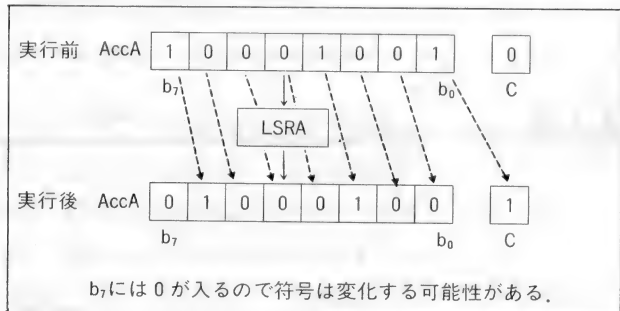
ASR命令と比較すると，全体が1ビット右にシフトする点は同じですが，b_7に0が入る点が異なります．そのためb_7が符号ビットのときには，その符号は無効になります．なお，Nフラグは常に0になっています．

ソース形式例　　LSR　；　LSRA　；　LSRB

MUL

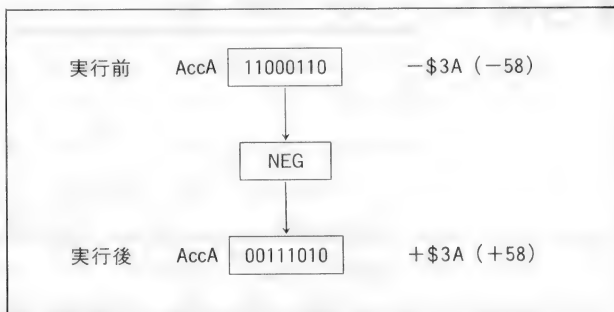
(MULtiply)

AccA, AccBにある符号なし2進数データ同士の乗算を行い，結果をアキュムレータDに格納します．

ソース形式例　　MUL

NEG

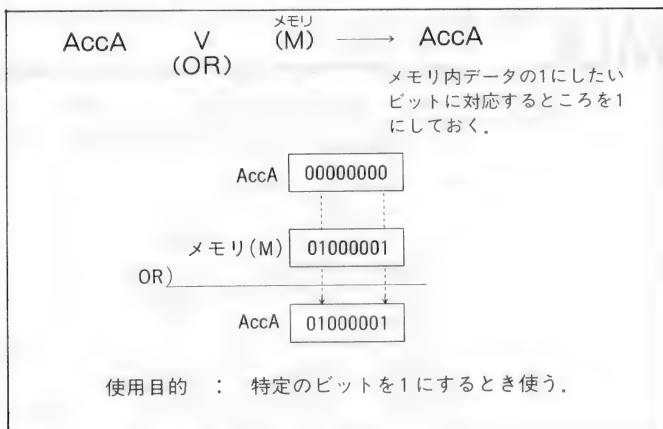
(NEGate)

アキュムレータまたはメモリの中のデータを，絶対値を変えずに符号を反転した数（2の補数）を作ります．たとえば5ならば－5になります．

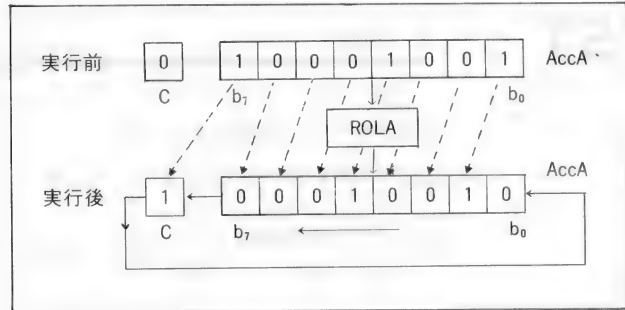
ソース形式例　　NEG　；　NEGA　；　NEGB

OR

(Inclusive memory into vgiuter)

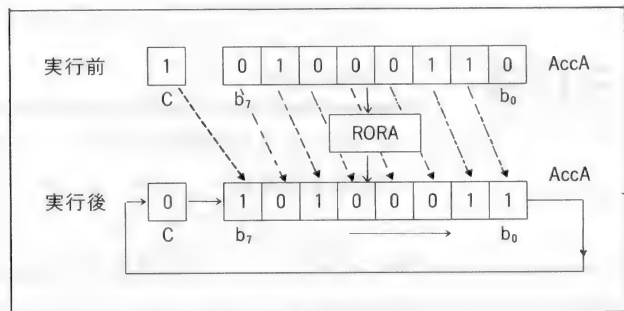
アキュムレータとメモリの内容との論理和を求めます．結果はアキュムレータに残ります．メモリ内のデータの特定のビットを1にするときに用います．

ソース形式例　　ORA　；　ORB

ROL (ROtate Left)

アキュムレータまたはメモリ内容を左方向へ1ビット回転させます．前述のＡＳＬ命令とは違って，b_7の内容はＣフラグを通してからb_0へローティートします．

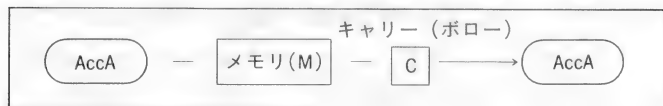
ソース形式例　　ＲＯＬ　；　ＲＯＬＡ　；　ＲＯＬＢ

ROR (ROtate Right)

アキュムレータまたはメモリの内容を右に1ビット回転させます．前述のＡＳＲ命令とは違って，b_0の内容はＣフラグを通してからb_7へローティートします．

ソース形式例　　ＲＯＲ　；　ＲＯＲＡ　；　ＲＯＲＢ

SBC (SuBtraCt with borrow)

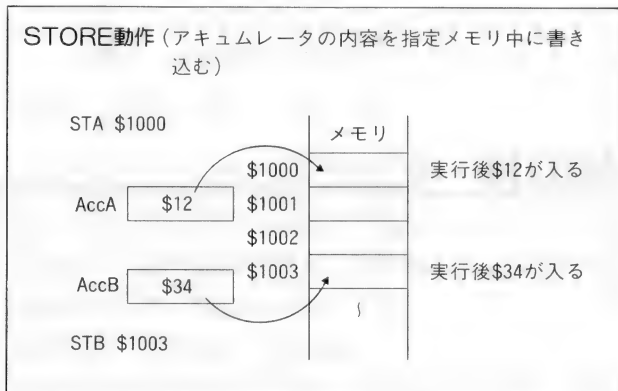
アキュムレータからメモリの内容を減算して（キャリーフ

ラグも含めて減算する)，結果をその指定レジスタに格納します．なお減算のときはキャリーをボローといいます．

ソース形式例　　ＳＢＣＡ　；　ＳＢＣＢ

ST

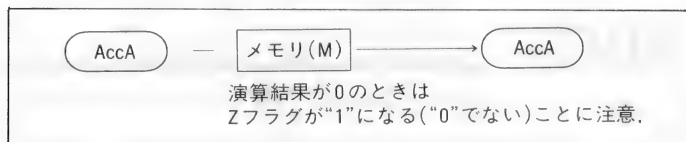
(STore)

アキュムレータの内容をメモリへコピーします．前述のＬＤ命令とは逆の動作をします．

ソース形式例　　ＳＴＡ　；　ＳＴＢ

SUB

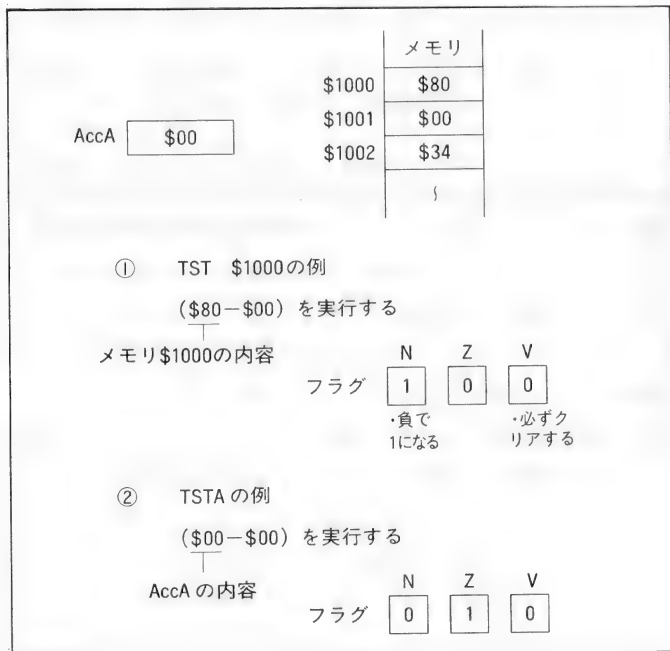
(SUBtract)

ＡＤＤ命令と逆の動作をします．アキュムレータからメモリの内容を減算処理します．Ｃフラグが１になった場合は，キャリーではなくボロー（借り）としての意味を持ちます．結果はアキュムレータに残ります．

ソース形式例　　ＳＵＢＡ　；　ＳＵＢＢ

TST

(TeST)

アキュムレータまたはメモリの内容を“0”と比較します．その結果N，Z，Vの各フラグが影響されますが，アキュムレータの内容は破壊されません．Nフラグはb_7ビットが1のときにセットされ，Zフラグはb_0～b_7の全ビットが0のとき1になります．Vフラグは常に0になります．単にフラグに影響を与えるだけの命令です．

ソース形式例　　ＴＳＴ　；　ＴＳＴＡ　；　ＴＳＴＢ

TFR

(TransFeR memory to momory)

8ビット・レジスタ同士でデータ転送をします．その結果，転送先のデータは転送元のデータと同じになります．EXG命令と似ていますが，異なります．ソース形式例のR1，R2は，R1レジスタの中身をR2レジスタに転送することを

意味します．なお，データ転送するレジスタ同士は同じビット長であることが必要です．

ソース形式例　　ＴＦＲ　Ｒ１，Ｒ２　（Ｒ１，Ｒ２＝Ａ，Ｂ，ＣＣ，ＤＰのうちのいずれか）

3 16ビット・レジスタ/メモリ命令

16ビット専用命令

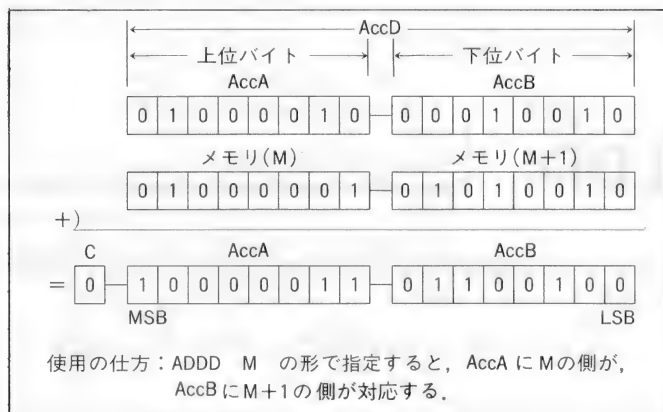
これまでの命令はすべて８ビット構成のレジスタやメモリ内のデータに作用するものばかりでしたが，このグループ内の命令は，命令の対象が16ビットのアキュムレータＤ（AccA，AccBを連結させたもの）や16ビット長のレジスタやデータに及ぶものです．ここで説明する命令は８種類です．

ADDD (ADD memory to D accumulator)

２バイトのメモリ内容を16ビットのアキュムレータＤに加算します．16ビット長を扱うときは上位バイト（b_{15} ～ b_8），下位バイト（b_7～ b_0）の配置関係に注意します．この命令による動作例は下図のようになります．

アドレッシングにおいては，上位バイトであるメモリ番地Ｍを示すと，下位バイトは自動的にＭ＋１番地が参照されます．下図の例では“ＡＤＤＤ　Ｍ”の様式で参照できます．

ソース形式例　　ＡＤＤＤ　Ｍ

使用の仕方：ADDD　M　の形で指定すると，AccA にＭの側が，AccB にＭ＋1の側が対応する．

CMPD

(CoMPare memory from D accumlator)

アキュムレータDと2バイトのメモリ番地Mの内容とを比較します．メモリの低いアドレスの方にデータの上位バイトを，高いアドレスの方にデータの下位バイトを入れます．動作はADDD命令の図と同様です．

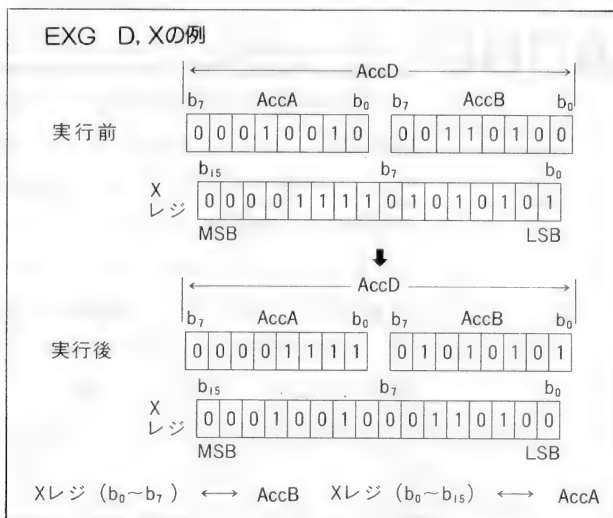
ソース形式例　　CMPD　M

EXG

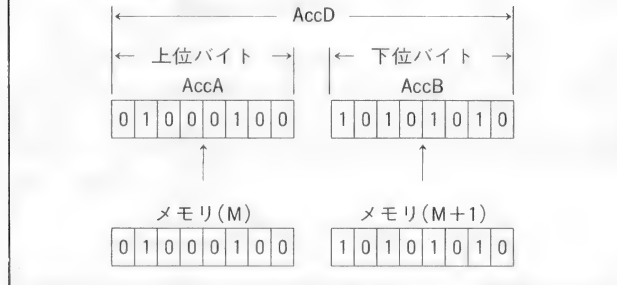
(EXchanGe D with X, Y, S, U,PC)

アキュムレータDと指定レジスタ同士がデータを交換し合います．

ソース形式例　　EXG　D，R（Rは16ビット・レジスタ）

LDD

(LoaD D accumulator from memory)

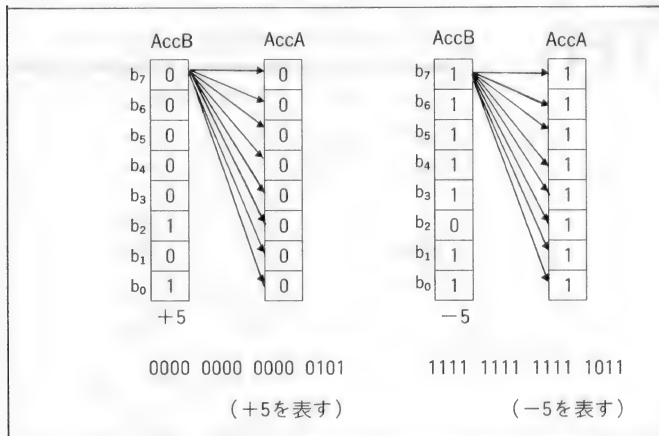
アキュムレータDに2バイト分のデータを格納します．このときのアキュムレータとデータの関係は下図のようになります．

ソース形式例　LDD　M

SEX

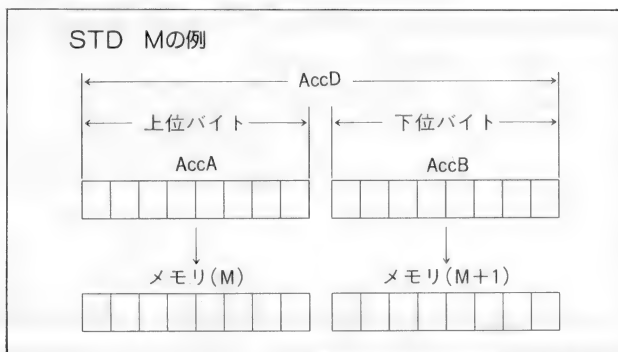
(Sign Extend)

符号（Sign）ビットに関する動作をします．アキュムレータBに入っている8ビットの符号付き2進数（b_7）を，アキュムレータAの全ビットにコピーし，16ビットに拡張します．8ビットの数値と同じ値の16ビット値を作るときに使います．

ソース形式例　　SEX

STD

(STore D accumulator to memory)

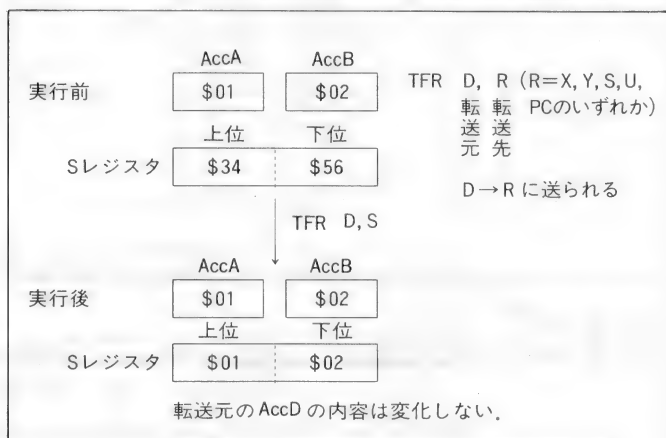
16ビット・レジスタのデータをメモリにしまいこむ（store）ためのもので，前述のＬＤＤ命令の逆の働きをします．

ソース形式例　　STD　M

SUBD (SUBtract memory from D accumulator)

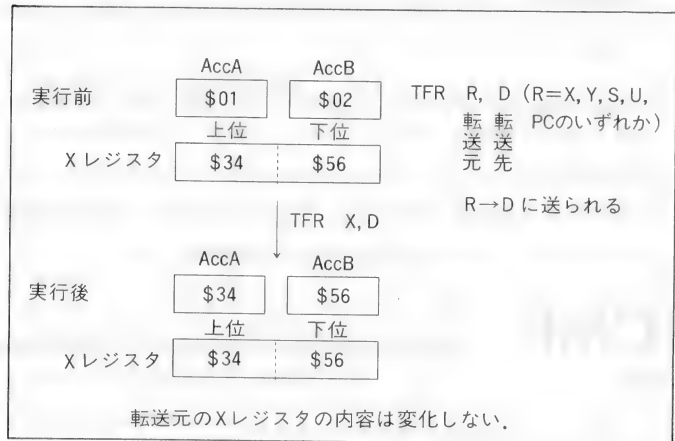
連続したメモリ番地M，M＋1の内容をアキュムレータDから減算します．M，M＋1とDレジスタとの関係はSTD命令の図と同じになります．結果はDレジスタに入ります．

ソース形式例　　SUBD　M

TFR (TransFeR registers to registers)

アキュムレータDの内容を，レジスタR（16ビット長のレジスタなのでD，X，Y，U，S，PCが指定できる）にコ

ピーする命令です．注意することはコピー先，元のレジスタは同一ビット長（この場合は16ビット長）のレジスタ同士であることです．

ソース形式例　　ＴＦＲ　Ｄ，Ｒ

以下の図は16ビットのレジスタの内容を，アキュムレータDにコピーする場合です．前述した“ＴＦＲ　D, R”の命令とは逆の動作をします．

インデックス・レジスタ／スタック・ポインタ命令

強力なインデックスト・アドレッシング

6809の命令の特長は，インデックスト・アドレッシングが大変強力なことです．このインデックスを生かした命令を使うことで，16ビット長の演算が簡単になります．機能としては，「2　16ビット・アキュムレータ／メモリ命令」で使用されたアキュムレータDの代わりに，インデックス・レジスタX，Y，スタックポインタ用レジスタS，U，プログラム・カウンタPCを使用する形態となります．ここで説明する命令は8種類です．

CMP (COMpare memory from registers)

16ビットレジスタS，U，X，Yからメモリ Mの内容データを差し引いて，その結果でフラグをセット，リセットします．8ビットのときと同じようにフラグのみを変化させます．

（M，M+1はメモリの番地を示す）

EXG

(EXcanGe regiters with regiters)

S，U，X，Y，D，PCのレジスタで，レジスタ同士がデータを交換する命令です．データはもちろん16ビットの内容です．

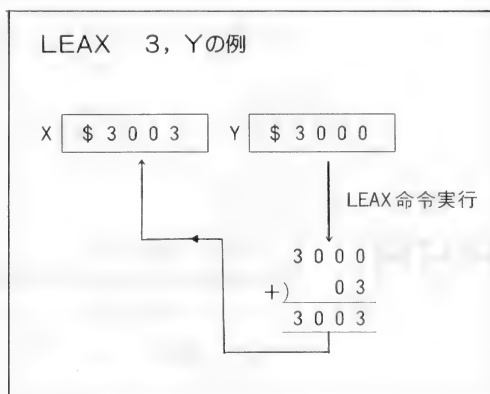
表記の仕方は“EXG　X，Y”のように記します．動作内容は交換データが8ビットから16ビットになったことを除けば，8ビットの場合の説明と同じです．

LEA

(Load Effective Address)

6809独特の命令です．ポインタ・レジスタであるS，U，X，Yに，実効アドレスを設定します．実効アドレスとは，命令が実行されるときに実際に参照されるアドレスまたはアクセスしようとするデータが存在するアドレスをいいます．この命令はあくまでも実効アドレスの値をレジスタに残すことが目的です．

結果を格納するレジスタとして，S，U，X，Yの4種類が指定できます．それぞれ“LEAS，LEAU，LEAX，LEAYと表記します．なお，このとき指定アドレスにアキュムレータDが入っていないことに注意してください．

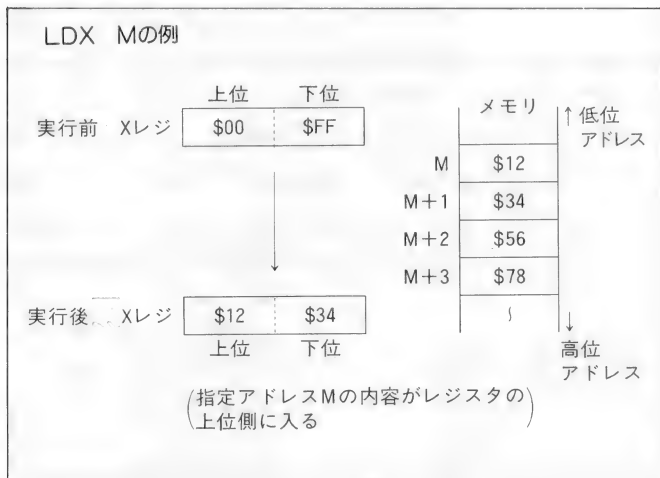
実際にLEA命令を使用したときの処理の流れは図のよう

になります．この図で，Xレジスタに入るデータには＄3003番地の中身ではなく，＄3003という値そのものが入ります．この命令の応用としては，次の命令表記ができます．

　　ＬＥＡＸ　１，Ｘ　　　Ｘレジスタの内容を＋１する
　　ＬＥＡＸ　－１，Ｘ　　Ｘレジスタの内容を－１する

LD

(LoaD registers from memory)

16ビットレジスタＳ，Ｕ，Ｘ，ＹにメモリＭ番地の内容を読み込みます．８ビットの命令の“ＬＤＡ，Ｂ”とは異なり，16ビット値（２バイト）データを扱います．データのメモリ上の位置関係に気をつけてください．

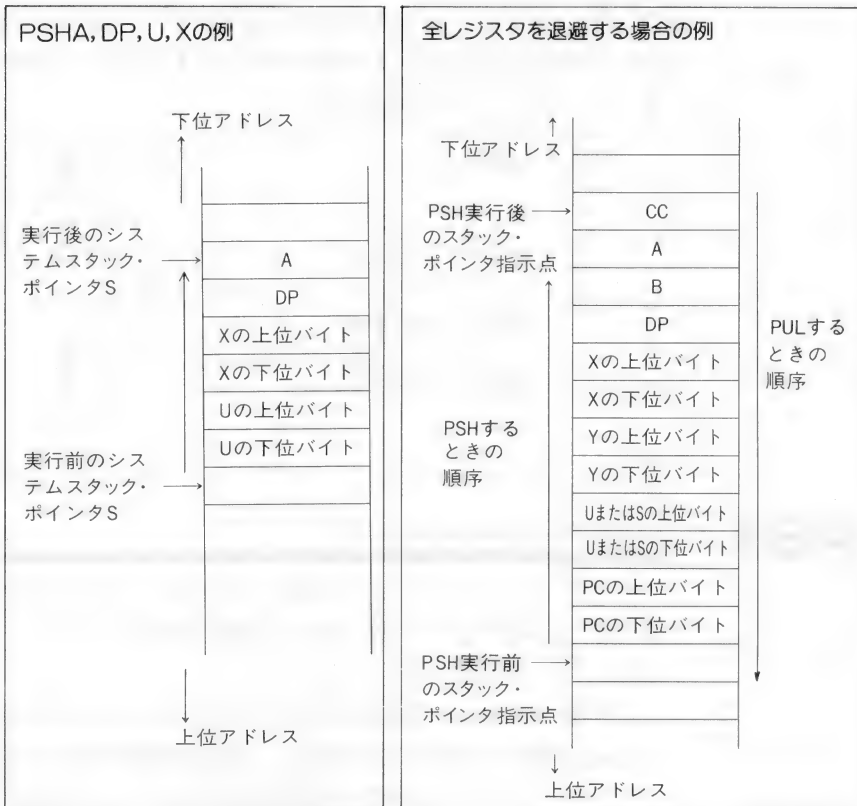
“ＬＤＸ　Ｍ”を実行したときの動作は図のようになります．この実行例ではＸレジスタについての例ですが，他のＳ，Ｕ，Ｙの場合も同様です．

PSH

(PuSH registers)

レジスタの内容をスタック・レジスタ（ＳとＵがある）に一時的に退避する命令です．したがってレジスタ内のデータを退避した後で，そのレジスタを使用して演算を行うことが

できます．演算が終わったら，先ほど退避しておいたデータを復帰し，また次の処理へ進みます．このように，ＰＳＨ命令を使用すれば１つのレジスタを多重に使用できます．表し方は，“ＰＳＨＳ　Ａ，ＤＰ，Ｕ，Ｘ”のように記します．

この命令を実行すると下図(左)のようになります．6800のときにはユーザがスタックに退避するレジスタの順序を指定するのが一般的でしたが，6809では退避するレジスタの順序が決まっていて，下図(右)のようになります。

最初にＰＣ，次にＵ，Ｙ，Ｘ，ＤＰ，Ｂ，Ａ，最後にＣＣをそれぞれ退避します．

PUL

(PUL registers)

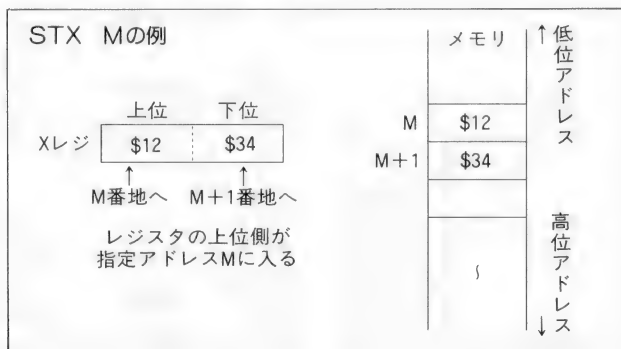
ＰＳＨ命令によってスタック・レジスタに一時的に退避した内容を，レジスタに復帰させる命令です．復帰するときは最初にＣＣが取り出され，次々にＰＳＨ命令のときと逆の順序でレジスタの内容が戻されます．

ST

(STore registers into memory)

16ビットのレジスタの内容をそれぞれメモリの中に格納する命令です．

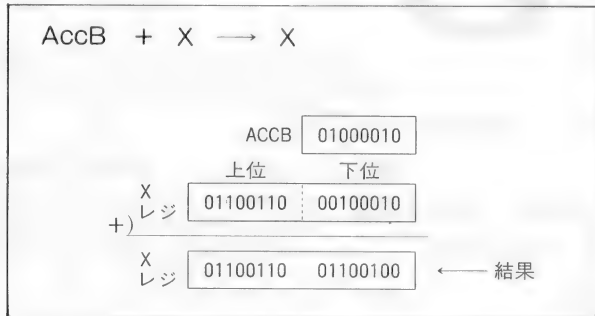
フラグＮ，Ｚが変化します（Ｖフラグはオーバーフローが生じないので“０”）．この命令の実行例は図のようになります．この図ではＸレジスタの例を取り上げましたが，ほかのＳ，Ｕ，Ｙについても同様です．

ABX

(Add B accumulator to X)

インデックス・レジスタＸの値にアキュムレータＢの値を符号なし２進数として加算します．演算の結果はインデックス・レジスタに残ります．

この命令はAccAでは実行することができません．あくまでAccB側だけに許されています．一般の加減算命令はメモリとアキュムレータの間で実行されるのですが，このＡＢＸ

命令だけはレジスタ同士で演算が可能です．ただし AccB は 8 ビットですから，1 回で演算できる数は 0 から255までの数に限られます．

実際にプログラムを組んでみると，インデックス・レジスタXに，ある数値を加えたい場合が 1 バイトの命令で実行できるので重宝な命令といえます．

ブランチ命令

プログラムの変更に必要な命令

プログラム実行の順序を変更する必要が生じたときに使用する命令群が6809には２種類用意されています．

１つはブランチ命令であり，もう１つはジャンプ命令です．この２つは明確に区別して考える必要があります．したがって，本書の分類法では本節でブランチ命令を説明し，「6　その他の命令」でジャンプ命令を説明することにします．

6809の命令の特徴はブランチ命令が豊富なことです．**表3-2**から理解できるように，ブランチ命令はほとんどの条件で分岐できます．この命令群のおかげ

図3-2　ブランチ命令の形式例

表3-2
ブランチ命令

ニーモニック・コード	動作内容
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid twos complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid twos complement result
BLT, LBLT	Branch if less than (signed)
BHI, LBHI	Branch if higher (unsigned)
BCC, LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

で，6809ではポジション・インデペンデントなプログラム（番地自由なプログラム）が自由に書けるわけです．

ブランチ命令は大きく分けると，**ショート・ブランチ**と**ロング・ブランチ**がありますが，命令名は主としてショート・ブランチの場合の名前を記します．ロング・ブランチの場合の名前は，ショート・ブランチの名前にLをつけます．

ショート・ブランチとロング・ブランチの違いは分岐できるアドレス範囲の差です．ショート・ブランチでは，分岐できる範囲は－128～＋127に限定されますが，ロング・ブランチの場合は＋32767～－32768の範囲が分岐の対象になります．しかし，命令構成はＢＲＡ命令の場合では，**図3-2**のように４バイト

構成になります．したがって安易にロング・ブランチを多用しますと，プログラムが大きくなり実行速度も落ちてしまいます．なお，オペランドの数値算出法は4章のリラティブ・アドレッシングの項で説明します．

項目はすべてショート名になっています．説明する命令は8種類です．

BEQ

(Branch if EQual)

第2章P53で説明したCC(コンディション・コード)と密接に関係します．CCのフラグの設定値と命令のニーモニック・コードの要求条件が一致したときに分岐します．ゆえに分岐命令の直前に条件判定のための演算が実行されている必要があります．BEQ命令は演算の結果，Zフラグが“1”のときブランチします．プログラム例を使って説明しましょう．

```
          :
         DECA          ………演算実行
         CMPA  #$03    ………フラグ設定
         BEQ   EXIT    ………ブランチ命令
          :
EXIT     LDA   #$FF
```

BNE

(Branch if Not Equal)

BEQ命令と反対の関係にあります．Zフラグが“0”で分岐します．

BMI

(Branch if MInus)

演算結果がマイナスのときに分岐します．ゆえにNフラグが“1”で動作します．

BPL

(Branch if PLus)

BMI命令の反対の働きをします．演算結果がプラスならばブランチします．Nフラグが“0”のときに動作します．

この命令を使えばビット7が1であるか0であるかの判定ができます．したがって，カナ文字かＡＳＣＩＩコードであるかの区別をするときに最適です．プログラムは次のようになります．

```
                 ⋮
        LDA     DATA……コード・データ読み込み
        BPL     ASC  ……条件分岐（ビット7＝0ならASCへ）
JIS     ANDA    #SFF
                 ⋮
```

BCS

(Branch if Carry Set)

演算の結果，Ｃフラグがセットされているときに分岐します．キャリーの有無の判定に使います．

BCC

(Branch if Carry Clear)

演算の結果，キャリーがクリアされているときに分岐します．ＢＣＳ命令とは反対の動作をします．

BVS

(Branch if oVerflow Set)

符号付き2進数での比較の結果(ここが大事です)，オーバーフローＶフラグ“1”で分岐します．

BVC

(Branch if oVerflow Clear)

ＢＶＳの反対の動作をします．符号付き2進数での比較の結果，オーバーフロー・フラグＶが“0”のときに分岐します．

以上のブランチ命令は，その条件判定がＣＣのビットの様子により分岐したわけですが，次からのブランチ命令は不等号条件も加わった判定で分岐できます．

8つの命令を紹介します．

BGT

(Branch if GreaTer)

G T は GreaTer の略で不等号の“＞”を意味し，フラグ条件は複雑です．フラグは“$Z \lor (N \oplus V) = 0$”のときです．これは，符合付き 2 進数の場合に，$Z = 0$ が第 1 条件で，$N = V$ が第 2 条件であることを意味しています．

BGE

(Branch if Greater than Equal)

G E は Greater than or Equal の略で，不等号の “≧” を意味します．フラグは “$(N \oplus V) = 0$” のときです．

BLE

(Branch if less than or equal)

L E は Less than or Equal の略で，不等号の≦を意味します。符号付き 2 進数で比較した結果が 0 よりも小さいか，または 0 に等しいときに分岐します。フラグは $Z \lor (N \oplus V) = 1$ のときです．

BLT

(Branch if Less Than)

L T は，Less Than で，不等号の “＜” を意味します．符号付き 2 進数での比較の結果 “0” よりも小さいときに分岐します．フラグは $N \fallingdotseq V$ のときです．

BHI

(Branch if Higher)

H I は，Higher の略で，不等号の “＞” を意味します．符号なし 2 進数としての比較の結果 “0” よりも大きいときに分岐します． フラグは“$C \lor Z = 0$”のときです．

BHS

(Branch if Higher or Same)

H S は，Higher or Same の略で，不等号の “≧” を意味します．符号なし 2 進数としての比較の結果，大きいかまたは等しいときに分岐します．フラグは “$C = 0$” のときです．

BLO

(Branch if LOwer)

ＬＯはLowerの略で，不等号の“＜”を意味します．符号なし２進数としての比較の結果，アキュムレータの値の方がメモリの内容よりも小さいことが条件です．フラグはキャリーが立った“Ｃ＝１”のときになります．機械語コードではＢＣＳ命令と同じです．

BLS

(Branch if LOwer or Same)

ＬＳはLower or Same の略で，不等号の“≦”を意味します．符号なし２進数としての比較の結果，アキュムレータの方がメモリの内容よりも小さいか同じであることが条件となります．フラグは“Ｃ∨Ｚ＝１”のときになります．

以上のブランチ命令のほかに，特殊なブランチ命令が３つあります．ＢＲＡ，ＢＲＮ，ＢＳＲ命令です．各々について説明していきます．

BRA

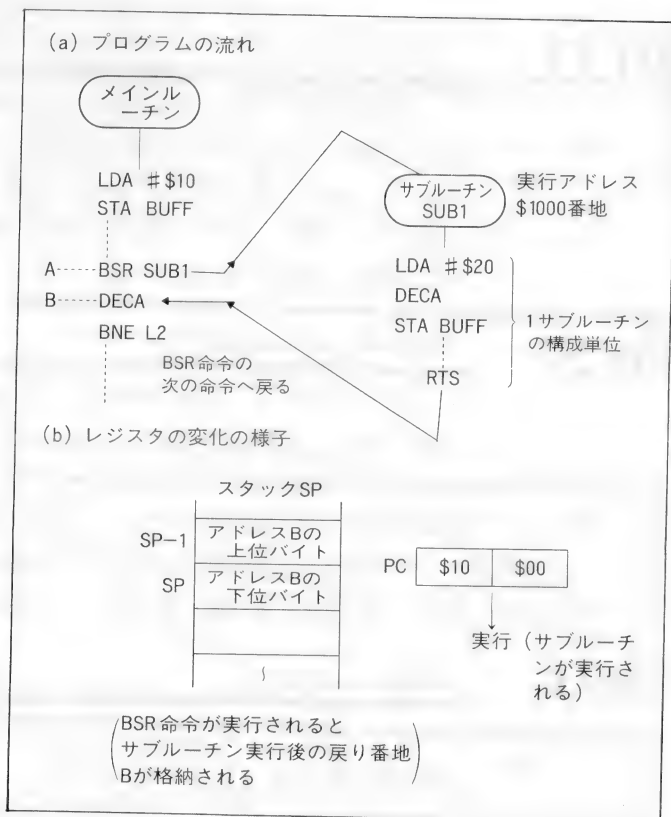
(BRanch Always)

現在のプログラムの実行順序を変更するときに使用する無条件分岐命令です．ＣＣ（コンディション・コード）がどのような状態になっていようとも分岐します．またフラグには何の影響も与えません．ＢＲＡは２バイト構成ですが，ＬＢＲＡ命令は３バイト構成です．

BSR

(Branch to SubRoutine)

サブルーチンに分岐する命令です．しかし分岐する際にＰＣ（プログラム・カウンタ）の内容を，スタックＳＰにプッシュしてから分岐先へ飛びます．後述のＪＳＲは絶対アドレスによる分岐タイプですが，ＢＳＲは相対アドレスによる分岐タイプです．実際にこの命令で実行される処理の様子を図(次ページ）に示します．

なお，ＬＢＳＲ命令は３バイト構成になります．ＬＢＲＡ

BRN

(BRanch Never)

とLBSRのみが3バイト構成で，ほかのロング・ブランチ命令はすべて4バイト構成になっています．

これはBRanch Neverの略で，BRA命令が無条件ブランチであったのに対し，この命令はフラグがどんな状態になっても決してブランチしません．2バイトのNOP命令と考えられます．またLBRNは4バイトのNOP命令に相当します．遅延ループの時間調整用やデバッグ時の不要命令をつぶすときなどに挿入すると大変有効です．

その他の命令

ちょっと変わっている命令

一般のメモリやレジスタを参照する性質の命令とは異なる命令の集まりです．その１つ１つが特徴を持った働きをして，プログラムやＭＰＵが効果的に動作できるように機能します．

ここで説明する命令は10種類です．

ANDCC (AND Condition Code registor)

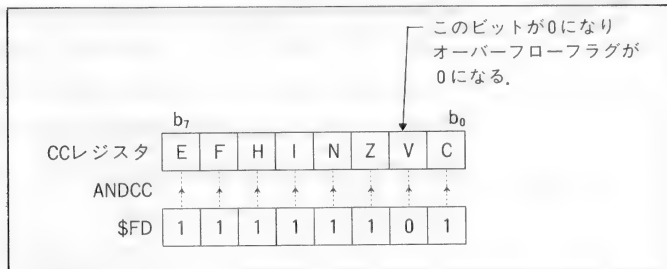
ＣＣ（コンディション・コード）の内容に，ある一定値（イミディエイト値）をＡＮＤしてＣＣのセット値を直接変更します．たとえばオーバーフローフラグであるＶフラグを０にするときは，

ＡＮＤＣＣ　＃＄ＦＤ

として実行します．理由は，下図を参照してください．この命令により，次のような擬似命令が作れます．

ＣＬＣ（Clear Carry）→ＡＮＤＣＣ　＃＄ＦＥ

Ｃフラブをクリアします．

ＣＬＩ（Clear Interrupt Mask）→ＡＮＤＣＣ　＃＄ＥＦ

ＣＬＶ (Clear Overflow) →ＡＮＤＣＣ　#＄ＦＤ

実は，上記の左の３つの命令は，6800にはあるのですが，6809の命令群の中にはありません．このようにＡＮＤＣＣ命令を使うことによって，6800の命令に等価な命令を作れるわけです．ゆえに6800のＣＬＣ命令に相当するものが必要なときには“ＡＮＤＣＣ　#＄ＦＥ”を使うことになります．

CWAI

(and Condition code register, then WAit for Interrupt)

割り込み待ちへの応答を早くするために作られた命令です．まず，オペランドのイミディエイト値（後述）とＣＣがＡＮＤされます．そしてＥフラグを立てて，ＳＰを除く全レジスタを退避させ次の命令の実行を中断し，割り込みの発生を待ちます．すでにレジスタが退避されているので，割り込み発生後すぐに割り込み処理を行うことができます．

ただし，割り込みからの復帰にあたってはＦＩＲＱからの復帰であっても，ＩＲＱからの復帰動作と同じように，ＳＰレジスタを除く全レジスタの復帰を行います．

イミディエイト値により，動作は次のように異なります．

ＣＷＡＩ　#＄ＥＦ　　ＩＲＱ割り込みを受け付ける
ＣＷＡＩ　#＄ＢＦ　　ＦＩＲＱ割り込みを受け付ける
ＣＷＡＩ　#＄ＡＦ　　ＩＲＱ，ＦＩＲＱ割り込みを受け付ける

なお，割り込みを待っている間，バスはハイインピーダンスになっていないので注意してください．

NOP

(No OPeration)

何もしない命令です．こんな命令が必要なのかと思う人もいるかと思いますが，実は大変有効な命令なのです．遅延時間発生ルーチン等における調整用時間の発生や，デバッグ作業などのときに不要な命令をつぶすなどの作業に大きな効果があります．

たとえばシステム・クロックが１MH_zのとき，この命令の

マシンサイクルは2なのでNOPの実行時間は2μsとなり，この時間だけスピードが遅くなります．したがって，必要時間数分NOP命令を続ければ所定の遅延が行えます．なお，時間調整用としてはBRN命令がマシンサイクル“3”，LBRN命令がマシンサイクル“5”なので，これをうまく組み合わせて目的の値になるようにします．

ORCC (OR Condition Code register)

前述のANDCC命令の逆の働きをします．CCの特定ビットをONにするのに多用されます．

JMP (Jump)

ブランチ命令のところでも述べましたが，ジャンプ（JMP）命令は本質的にブランチ命令とは性質が異なっています．ブランチ命令系が相対アドレス値をオペランドに持って実行されるのに対し，JMP命令はオペランドが絶対アドレスなのです．

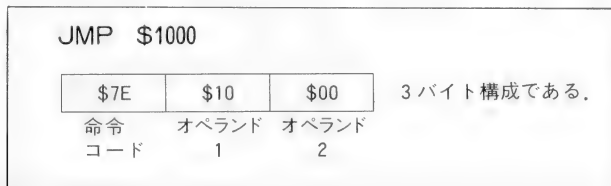
命令コードは＄7Eですので，もし＄7000番地にジャンプさせたいときは“＄7E　＄10　＄00”と3バイト並べることにより，実行させることができます．飛び先番地が直接オペランドから読み取れるので理解しやすいのですが，6809の機能を生かそうとしたなら，なるべくこのような絶対番地形式の命令は使わないようにします．できるだけ相対アドレスでプログラムを書くように努力した方がよいでしょう．

JMP命令の基本形態は図に示すとおり3バイト構成になります．

JMP　$1000

$7E	$10	$00
命令 コード	オペランド 1	オペランド 2

3バイト構成である．

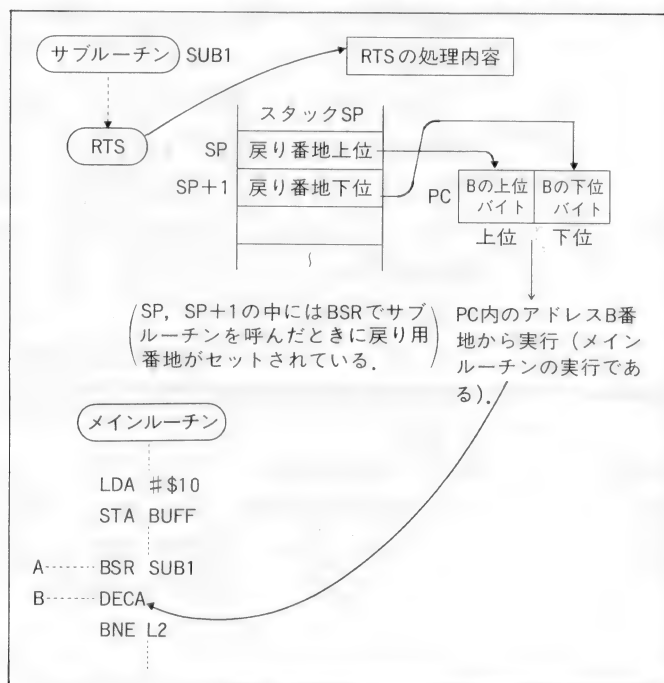
JSR

(Jump to SubRoutine)

先のＢＳＲ命令がオペランドに相対アドレス値を持って分岐するタイプでした．ＪＳＲ命令はオペランドに絶対アドレス値を持つことを除けば，動作はまったく同じです．やはりサブルーチンに飛びますので，サブルーチン側ではＲＴＳ命令でプログラムを終了しておけば自動的にサブルーチン実行終了後，ＪＳＲ命令の次の命令があるアドレスに戻ってくることができます．

なお，この命令もＪＭＰ命令と同じく，できるだけ相対アドレス形式のＢＳＲ（またはＬＢＳＲ）を使用した方が6809らしいプログラムを書くことができます．

RTS

(ReTurn from Subroutine)

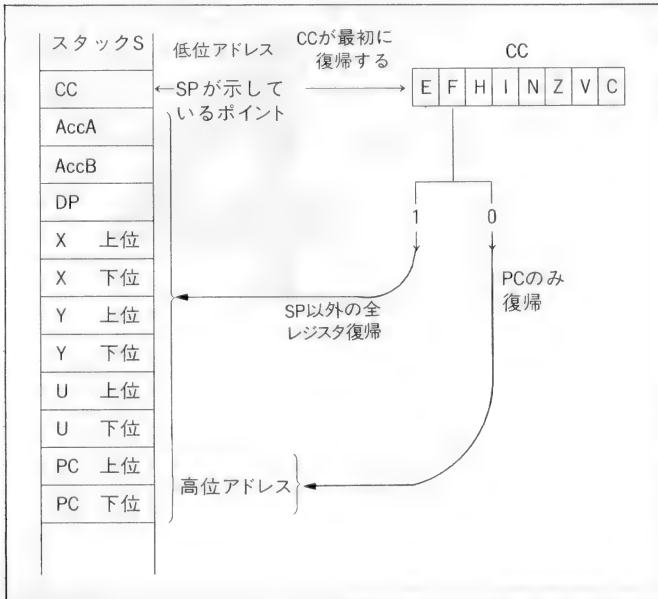
サブルーチンのプログラムは必ずこの命令で終了する必要

があります．ＪＳＲ，ＢＳＲ，ＬＢＳＲで呼ばれたサブルーチンを実行後に，元の呼んだ方のメインルーチンに戻るための処理を行ってくれます．スタック・レジスタＳＰには，サブルーチンからの戻り先番地が格納されていますので，戻り先番地を取り出しPCにセットします.そしてＰＣが示す番地に飛ぶことにより,元のメインルーチンに戻ることができます.

RTI

(ReTurn from Interrupt)

割り込み処理を終了させてメインルーチンに戻るための処理を行う命令です．この命令を割り込み処理ルーチンの最後に書いておけば，割り込みルーチンの実行終了後自動的にメインルーチンに戻ることができます.

ＲＴＩ命令は最初にスタックからＣＣを復帰させます．ＣＣ内のＥフラグが“１”のときは，次にAccA，AccB，X，Y，Ｓ/Ｕ，ＰＣを復帰させます．Ｅフラグが“０”のときにはＰＣのみ復帰させます.

SWI

(SoftWare Interrupt)

割り込み発生のための命令です．ＳＷＩ，ＳＷＩ２，ＳＷＩ３があります．ＩＲＱ，ＦＩＲＱ，NMＩの割り込みは，ＭＰＵの端子がハード的に“Ｌ”になって行われますが，この命令を使えばソフト的に割り込みが発生します．プログラム内で本命令に出会うと，それぞれの割り込みベクタの示す番地へ処理が移ります．

SYNC

(SYNChronize with interrpt line)

この命令を実行すると，ＭＰＵは次の命令の実行を中断して割り込みの発生を待ちます．割り込みが発生すると，次の命令を実行します．ただし割り込みピンが３マシンサイクル以上“Ｌ”になって，割り込みがマスクされていないときは割り込み処理に入ります．とくにフロッピーディスクからのデータ読み取りのように，データを高速で処理するときに力を発揮します．ＳＹＮＣ命令で待機している間はバスがハイインピーダンスになっているので，DMA（ダイレクト・メモリ・アクセス）を使い，データは外部から直接にメモリ内に取り入れることが容易にできます．

Chapter Four

アドレッシング形式と割り込み

MC6809の命令形態がほかの8080やZ-80等と大きく異なっている点は，アドレッシングモードの豊富さにあります．その豊富な命令によってレジスタ同士がお互いに修飾しあって多様な命令形態を作っています．

アドレッシングモードは，最初から全部のモードを理解しようとせず，まず自分の試してみたいモードを集中して，理解するのが効果的です．

第3章の解説では各命令の機能について説明しましたが，本章ではアドレッシングモードの違いによって，各命令の機能がさらに飛躍的に高まっていく様子を示していきます．

また6809MPUの性能が評価される理由として，割り込み機能が豊富なことも忘れてはなりません．7種類の割り込みが用意されていますので，ほとんどの割り込み処理がMPUのみで処理できます．マイクロコンピュータを用いて効果的な制御機器を製作するならば，割り込みを用いることは絶対必要です．

本章では割り込み動作の基本的概念と6809MPUの割り込み機能から説明していきます．

アドレッシングモード

命令の構成,オペランドポストバイト

一般にコンピュータの命令語は命令部とオペランド部から成り立っています（図4-1）．命令部は命令の種類を示し，オペランドは命令の対象となるもので，アドレス情報等を与えます．

命令の基本形態は簡単ですが，実際はこの形態のバリエーションが数多く生じます(図4-2)．このような変化は，命令コードとオペランドが図4-2のように変化することで生じます．ＭＰＵはおのおの命令の長さを命令コードによって知り，その情報によってオペランドを必要なバイト数だけ取り込みます．オペランドも長さと種類によって変化しますが，そのオペランドの変化を表すのがアドレッシングモードです．アドレッシングが豊富なことは6809ＭＰＵの特長の１つです．

例をあげて説明しましょう　たとえば“ＬＤＡ”というＬＤ命令の場合を考えてみます．

このＬＤ命令は「アキュムレータＡに，メモリ内のデータを転送しなさい」という命令です．しかし，そのデータの内容あるいはどこの場所からという記述がありません．すなわち，アドレッシングモードはそのデータの内容あるいはどこの場所からという記述を補う形式といえます．

図4-1　命令語の構成

命令部	オペランド部

・命　令　部……インストラクション・コード
　　　　　　　　オペレーション・コード，オペコード
・オペランド部……オペランド

図4-2 命令語の構成形態

(1)命令コード(オペコード)の長さが1のとき

※1コマが1バイトを表わす

①	オペコード	オペランド			
②	オペコード	オペランド			
③	オペコード	オペランドⒽ	オペランドⓁ		
④	オペコード	ポストバイト			
⑤	オペコード	ポストバイト	オペランド		
⑥	オペコード	ポストバイト	オペランドⒽ	オペランドⓁ	

(2)命令コード(オペコード)の長さが2のとき

⑦	オペコードⒽ	オペコードⓁ			
⑧	オペコードⒽ	オペコードⓁ	オペランド		
⑨	オペコードⒽ	オペコードⓁ	オペランドⒽ	オペランドⓁ	
⑩	オペコードⒽ	オペコードⓁ	ポスト・バイト		
⑪	オペコードⒽ	オペコードⓁ	ポストバイト	オペランド	
⑫	オペコードⒽ	オペコードⓁ	ポストバイト	オペランドⒽ	オペランドⓁ

アドレッシングモードには6つの種類があります．アドレッシングモードの中にはオペランドの前にある記号で区別しているものもあります． アドレッシングモードと区別するアドレッシングモードの記号は次のようになります．

- **エクステンデット・アドレッシング** → “>”
- **ダイレクト・アドレッシング** → “<”
- **イミディエイト・アドレッシング** → “#”
- **インヘレント・アドレッシング**
- **インデックスト・アドレッシング**
- **リラティブ・アドレッシング**

なお，インデックスト・アドレッシングについては，さらに複数のタイプがあります．

1・1　アドレッシングモードの種類

アドレッシングモードは6つの種類があります．以下に簡単に説明していきます．

◉エクステンデット・アトレッシング(Extended Addressing)

アドレスの16ビット（2バイト）を直接オペランドに記述するタイプのモードです．エクステンデッドとは「拡張された」という意味で，アドレス値を2バイトで拡張表現するためにこの名前がつけられました．ここではＬＤ（ロード）命令の場合をサンプルとして取り上げて説明してみます．

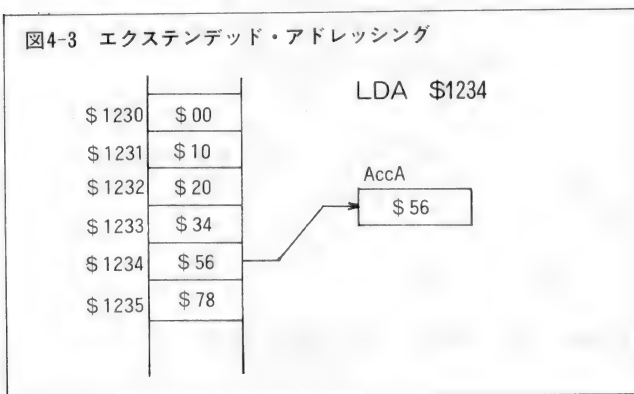
今，＄1234（＄は16進数のこと）番地に＄56というデータが入っていたとします．そのデータをアキュムレータA(AccA と略す)に格納するための命令は“ＬＤＡ　＄1234”と書いて実行すると，AccA の中には16進数の56が入ります（図4.3 ）．要するにオペコードのすぐ後にあるオペランドの2バイトが実効アドレスとなり，そのアドレスの中身が AccA に入るわけです．

図4-2 の命令語の形態でいうと，③のケースです．

ニーモニック（操作記号）による表現は，

ＬＤＡ　＞＄1234
ＬＤＡ　＄1234
ＬＤＡ　＞ＤＡＴＡ

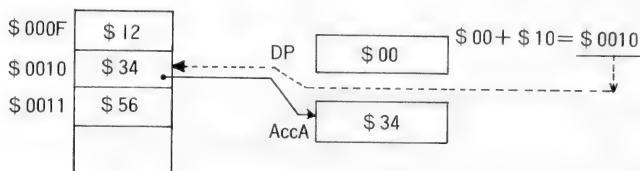
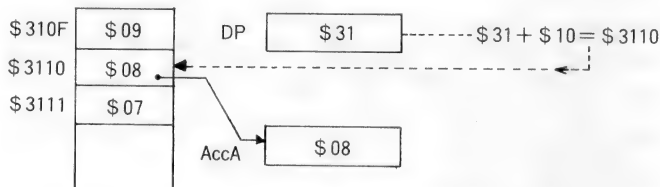
のように表記します（＞は省略可能です）．

図4-3　エクステンデッド・アドレッシング

◉ダイレクト・アドレッシング(Direct Addressing)

アドレスの上位８ビットをＤＰ（ダイレクト・ページ・レジスタ）から与え，下位８ビットだけをオペランドで指定するモードです．8080やZ-80にはない6809特有のものです．

6809にはＤＰという８ビットのレジスタがあり，プログラムによってＤＰに任意の数値を入れることができます．ダイレクト・アドレッシングは，このＤＰを利用して16ビットのアドレスを指定できるアドレッシング方式です．

この方式ではエクステンデット・アドレッシングに比べて，命令が１バイトだけ短くてすむので，実行を処理する時間は短くなります．先のエクステンデッド・アドレッシングが３バイトのマシン語となるのに対し，このモードでは２バイトのマシン語に変換されます．オペコードの後の１バイトがアドレスの下位バイトになり，ＤＰの内容が上位アドレスとなった２バイトの実効アドレスを参照します．モード例は**図4-4** のようになります．

図4-4　ダイレクト・アドレッシング

(a) DPレジの中身＋$10＝実効アドレス
＝$0010

(b) DPレジの中身＋$10＝実行アドレス
＝$3110

6800ではＤＰが強制的に“０”の形態になりますが，6809は64Ｋの空間にＤＰを設定します（上位バイトが00～ＦＦまでの間）．このモードにおける＄0000～＄00ＦＦまでの間のゼロページは，主にシステムが使用する形態が一般的です（68系では上位バイトが00のメモリ範囲をゼロページと呼ぶ）．

ニーモニックで表示するときは，

　ＬＤＡ　＜ＢｕＦ（ＢｕＦは記号番地である）

　ＬＤＡ　＜＄10

　ＬＤＡ　＄10

のように表記します．

“＜”記号はダイレクト・アドレッシングであることを表す記号ですが，オペランドが１バイトのときにはこの記号がなくてもダイレクト・アドレッシングと判断します．

◉イミディエイト・アドレッシング(Immediate Addressing)

命令コードに続くオペランドの値をデータとして用いるモードです．そのためにアドレスの記述がありません．

定数を扱うときに便利で，オペランドに書いた値はイミディエイトに（即座に）参照できます．オペランドの値そのものがデータとして用いることができるので，速度の向上につながります．データの長さはアキュムレータが８ビットですので，やはり１バイトの長さに限られます．モード例は**図4-5**のようになります．

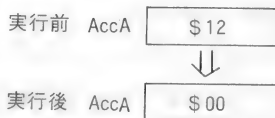
ニーモニック表示は，“ＬＤＡ　#＄10”のように表します．“#”はイミディエイトを表す記号です．

図4-5　イミディエイト・アドレッシング

◉インヘレント・アドレッシング(Inherent Addressing)

オペランドにアドレスの値やデータを持たないで，命令コード１バイトだけで形成されるモードです．オペランドを必要としないのは，ＣＬＲＡとかＣＯＭＢなどの命令を実行するのに必要な情報がオペコードの中に含まれているからです．

命令長は他の命令形式と比べてもっとも短く，実行スピードも速くなっています．モード例は**図4-6** のようになります．

図4-6　インヘレント・アドレッシング

◉インデックスト・アドレッシング(Indexed Addressing)

この命令語形式は，6809の大きなセールスポイントになっているほどすばらしい機能を持っています．このモードを上手に使えるか使えないかでプログラムのできぐあいが大きく左右されます．

このインデックスト・アドレッシングは多くの種類があり，その構成も複雑で，そのすべてを本項で述べることは不可能です．本来ならプログラム例を示しながら解説するべきでしょうが，スペースの関係で，今回はできるだけ簡略にしました．

本書ではこのモードを大きく分けて６種類あります．６種類の形態については次節に譲るとしまして，ここでは新たに使うポストバイトという用語について解説します．このモードで使用するオペコードは，巻末にあるインデックスト・モードのオペコード表を使います．さらに**表4-1**のポストバイトも使用します．

＊ポストバイトについて

今まで命令語の形式は**図4-7** のいずれかの形式で表されていましたが，ここでは新しく**ポストバイト**という新しい概念を使います．ポストバイトの形式は**図4-8** のようになります．

このポストバイトの採用によって，命令の表現法が飛躍的に豊富になります

表4-1 インデクスト・アドレッシングモード用のデータ

タイプ	形式	ノン・インダイレクト				インダイレクト			
		アセンブラ形式	ポストバイト・オペコード	+~	+#	アセンブラ形式	ポストバイト・オペコード	+~	+#
コンスタント・オフセット	No Offset	, R	1RR00100	0	0	[, R]	1RR10100	3	0
	5 Bit Offset	n, R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
アキュムレータ・オフセット	A-Register Offset	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
	B-Register Offset	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	D-Register Offset	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
自動インクリメント デクリメント	Increment By 1	,R+	1RR00000	2	0	使用不可			
	Increment By 2	,R++	1RR00001	3	0	[,R++]	1RR10001	6	0
	Decrement By 1	,−R	1RR00010	2	0	使用不可			
	DeCrement By 2	,−−R	1RR00011	3	0	[,−−R]	1RR10011	6	0
プログラム・カウンタ・リラティブ	8 Bit Offset	n,PCR	1XX01100	1	1	[n,PCR]	1XX11100	4	1
	16 Bitt Offset	n, PCR	1XX01101	5	2	[n, PCR]	1XX11101	8	2
エクステンデットのインダイレクト	—	—	—	—	—	[n]	10011111	5	2

R=X, Y, UあるいはS　　RR : 00=X　01=Y　10=U　11=S
X=ドント・ケア　　※+~と+#は付加するサイクル数とバイト数を表す

図4-7 命令話の形式（オペコードが1バイトの場合）

オペコード

オペコード	オペランド

オペコード	オペランド上位	オペランド下位

図4-8 ポストバイト・オペコードの形式例

オペコード	ポストバイト

□ 1つが8ビット構成

オペコード	ポストバイト	オペランド

オペコード	ポストバイト	オペランド上位	オペランド下位

表4-2　ポストバイト

(a)

レジスタ名	2進コード
D	0000
X	0001
Y	0010
U	0011
S	0100
PC	0101
A	1000
B	1001
CC	1010
DP	1011

(b)

ニーモニック	アドレッシングモード			操作内容
	レジスタ			
	OP	~	#	
TFR R1, R2	1F	7	2	R1→R2
EXG R1, R2	1E	8	2	R1↔R2

が，逆にそれだけ使いこなすようになるには大変です．

ポストバイトは，オペランドと異なり，命令の表現力を豊富にするための補助句です．“ＴＦＲ　Ａ，ＤＰ”と書きますと，これは「Ａレジスタから ＤＰレジスタへ」データを移送することを表しますが，この命令のマシン語を見てみますと，１Ｆ　８Ｂ　となっていますが，８Ｂがポストバイト，１Ｆがオペコードとなります．

ポストバイトを使う場合は限定されています．ＴＦＲ，ＥＸＧ，ＰＳＨ，ＰＵＬの４つの命令とその他の命令でインデックスト・アドレッシングモードの場合だけになります．

ＴＦＲ，ＥＸＧ命令は１つのグループであり，そのポストバイトは**表4-2**(a)のようになります．この表を用いて“ＴＦＲ　Ａ，ＤＰ”をマシン語に変換してみましょう．

ＴＦＲ命令を命令表で見ると，**表4-2**(b)のようになっていますから，オペコードは１Ｆであることがわかります．次に**表4-2**(a)から,Ａレジスタは1000，ＤＰレジスタは1011がそれぞれの表現コードであることがわかります．

結局，マシン語は	オペコード	転送元（上位4ビット）	転送先（下位4ビット）
	１Ｆ	1000	1011
		↓	↓
		＄８	＄Ｂ

より“１Ｆ　８Ｂ”となります．

◉リラティブ・アドレッシング(Relative Addressing)

このアドレッシングの充実が80系ＣＰＵと大きく異なる部分です．このモードは前に述べたブランチ命令に使われ，リロケータブルなプログラムの作成ができます．

命令の形式はポストバイトはなく，オペコードとオペランドだけで構成されています．アドレッシングで使用されるアドレスはメモリに１つずつつけられた絶対アドレスを使わないで，その命令から何番地離れているかという相対アドレス値を使います． 次のプログラム例を見てみましょう．

```
アドレス  ニーモニック
$100      LDA  #$10
$102      STA  $2000
$105      JMP  $100
```

このプログラムは＄100 から配置されていますが，もしこのプログラムを＄200 から配置したいとすると，次のように書き換える必要が生じます．

```
アドレス  ニーモニック
$200      LDA  #$10
$202      STA  $2000
$205      JMP  $200 ←──書き換えている
```

しかし，このような方法ではアドレスが変更されるたびにプログラムを書き換えることになり，とても非能率といえます．相対アドレスを採用すればこの面倒な点は解決されます．例を次に示します．

```
アドレス  ニーモニック
$100      LDA  #$10
$102      STA  $2000      $100 へ飛ぶ
$105      BRA  $F9
```

このプログラムを＄200 から配置してみますと，次のようになります．

```
アドレス  ニーモニック
$200      LDA  #$10
$202      LDA  $2000
$205      BRA  $F9
```

プログラムは同じものです．注意することは，ＢＲＡの次のオペランドのオフセット値が絶対アドレスでないことです．このオフセット値は，ＢＲＡという命令から飛び先が何番地ずれているかを示す数値です．これを求める手順を次に示します（＄100番地から配置されている場合）．

＄105 ＋2＋x＝＄100　になるxを求めればよいのです（＋2するのは，ＢＲＡが2バイト命令なので実行後のアドレスは＋2されているから）．

x＝＄100 －＄105 －＄2＝＄100 －＄107 ＝－＄07

－＄07 を2の補数で表現しますと，次のようになります(下位2バイトのみが計算に使われる)．

＋＄07 ＝ 0000 0111　　－＄07 ＝ 1111 1001

1111 → $F，1001 → $9，$F9

前の例はブランチ先が元に戻る方向（負の方向）の場合でした．では，ブランチ先が前進方向（正の方向）へ飛ぶ場合の例ではどうでしょうか．次の計算手順を見てみましょう．この例はＢＲＡの飛び先が＄108 番地になるような××を求めています．

```
$100  LDA  #$10
$102  STA  $2000
$105  BRA  ××   ─┐
$107  CLRA       │ $108 へ飛びたい
$108  LDA  #$20 ←┘
```

計算してみましょう．

＄105 ＋2＋x＝＄108

x＝＄108 －＄105 －＄2＝＄08－＄07＝＄01

これでオフセットは01となりますので，前のリストは次のようになります．

```
$100  LDA  #$10
$102  STA  $2000
$105  BRA  $01   ←オフセット値
$107  CLRA
$108  LDA  #$20
```

これまで説明してきた内容は，ショート・リラティブの例です．ショート・タイプでは

負方向に－128　（＄80）

正方向に＋127　（＄７Ｆ）

だけ飛ぶことができます．飛び先番地の計算結果のチェックは簡単で，次のようになります．

・もし負方向へのオフセット値が00～＄７Ｆの間の数になったときは誤まり

・もし正方向へのオフセット値が＄80～＄ＦＦの間の数になったときは誤まり

以上はショート・リラティブ・アドレッシングの例です．ほとんどのプログラムはショート・リラティブで処理できますが，64Kバイトの中を自由に相対アドレスで飛び回りたいというときはオフセット値を－32768 から＋32767 まで拡張したロング・ブランチ・リラティブ・アドレッシングを利用します（ＬＢＲＡ，ＬＢＳＲのようにLong の意味のＬが頭についています）．

1･2　インデックス・モードの種類

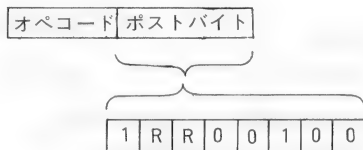
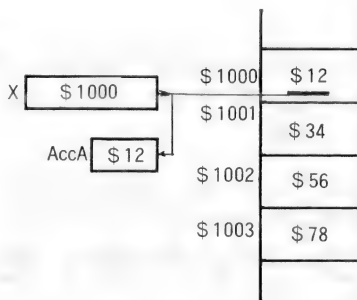
これまで述べたアドレッシングはレジスタに対するものだったり，メモリについても固定されたアドレスに対してだけ動作するものでした．したがって複数桁のデータを操作する場合にはとても不便でした．このモードはその問題を一挙に解決します．形態は６種類あります．

(1)オフセットなしインデックスト・アドレッシング

今までオペランドにアドレスの値を書き込んでオペコードに渡していましたが，この命令形態ではアドレスの値を前もってレジスタ（インデックス・レジスタXなど）に渡しておき，そのレジスタを参照することで実行します．

たとえば，“ＬＤＡ　，Ｘ”は，Ｘレジスタの示すアドレスからデータを読み出し，AccA にロードすることを意味します．Ｘレジスタには前もってアドレス値をロードしておきます．一度Ｘレジスタにロードしておけば，その後は何回でも使うことができます．ここで，どのレジスタを使うかによって，ポストバイトが活用されます．ニーモニックの記法は次のようになります．

命令コード	定数またはアキュムレータ名	ポインタレジスタ名

図4-9　オフセットなしインデックスト・アドレッシング

この形式で，さっきの命令を書き直すと，“ＬＤＡ　，Ｘ”と書くことができます．命令形態は図4-9 のようになります．実際に命令を書いてみましょう．

“ＬＤＡ　，Ｘ”をマシン語に直してみます．

巻末のオペコードの表中のソースコードＬＤＡのところを，横へ左から右に見ていきます．アドレッシングモードのインデックスト・アドレッシングの項でＯＰはＡ６になっています．このＡ６がオペコードです．

次に#の部は２＋になっていますが，この＋がポストバイトがあることを示します．したがって**表4-1** を参照して，ポストバイトを決めます．タイプはコンスタント・オフセットでゼロオフセット，非間接（ノン・インダイレクト）で，ポインタ・レジスタはＸなのでポストバイトは次のように構成されます．

１００００１００
Ｘ指定

結局，命令は次のようになります．つまり“＄Ａ６，＄84”の２バイト構成になります．**図4-9**(b)に実際のデータの流れを示します．

(2)定数オフセット付きインデックスト・アドレッシング

(1) のオフセットなしインデックスト・アドレッシングのところの“0”というオフセットを，ある一定値にしたものがこのモードです．

オフセット値により，

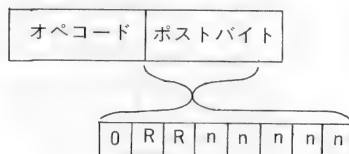
　5 ビット　　−16〜＋15
　8 ビット　　−128 〜＋127
　16ビット　　−32768 〜＋32767

という 3 クラスが指定できます．

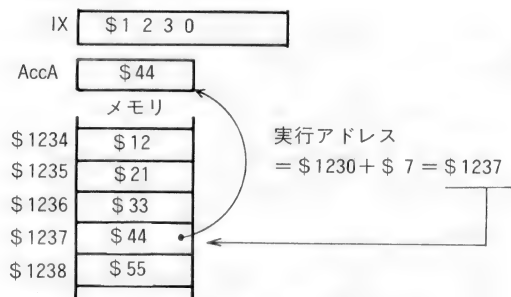
注意する点はオフセットが「符号付き 2 進数」ということで，最上位ビットが符号（0 なら正，1 なに負）を表します．5 ビットオフセットの場合を**図4-10**(a) に示します．

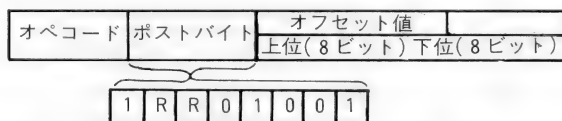
ではマシン語を**表4-1** から構成してみましょう．**表4-1** より，5 ビットオフセットの場合に，ポストバイトは“0 R R nnnnn ”となります．この nnnnn はオフセット値を符号付き 2 進数で表したものなので 7 を 2 進数で表し“00111 ”とします．R Rは X つまり00ですから，ポストバイトは00000111となります．16進数読みでは＄07です．結局，マシン語は“＄A 6　＄07”となります．この

図4-10　定数オフセット付きインデックスト・アドレッシング

LDA　7, Xの例

・RRの値は
IX＝00　IY＝01　US＝10　SP＝11　の
値をそれぞれとる．本例では00となる．
・ nnnnn の値はオフセット値で 5 ビットの補数表現であるので nnnnn ＝00111 となる．

(a) 5 ビット・オフセットの例

図4-10　定数オフセット付きインデックスト・アドレッシング

(b) 5ビット・オフセット例

(c) 8ビット・オフセット例

(d)16ビット・オフセット例

過程を図4-10(b) に示します.

8ビット・オフセットのときはオペコード，ポストバイト，オフセット値の3バイト構成です. 16ビット・オフセットのときはオペコード，ポストバイト，オフセット値上位，オフセット値下位の4バイト構成になります.

8ビット，16ビット・オフセットの構成はそれぞれ図4-10(c)，図4-10(d) のようになります.

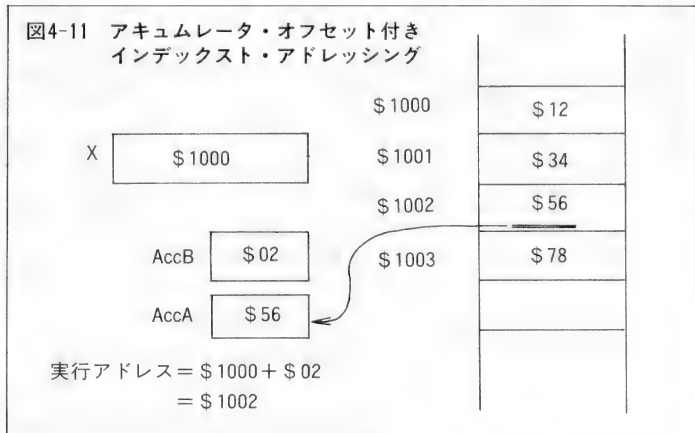
(3)アキュームレータ・オフセット付きインデックスト・アドレッシング

(2)のモードのオフセット値は定数でしたが，オフセット値をアキュムレータの内容にしています．ポストバイトはアキュムレータにより異なります．

AccA オフセット　　1 R R00110
AccB オフセット　　1 R R00101
AccD オフセット　　1 R R01011

アキュムレータ・オフセットの場合，AccA，AccB を指定すると 8 ビットのオフセットと解釈し，Dを指定した場合は16ビットのオフセットとして解釈されます．

たとえば，“LDA　A，X”では，AccA 使用時のマシン語は“＄A 6，＄86”となります．このアドレッシングの処理の流れは**図4-11**のようになります．

図4-11　アキュムレータ・オフセット付きインデックスト・アドレッシング

(4)自動増減型インデックスト・アドレッシング

このモードは基本的にはオフセットなしインデックスト・アドレッシングの一種です．

たとえば“LDA　，X”で考えてみます．この形式で，実行後または実行前にXの値が＋1か－1してくれたら非常に便利になります．6809はこの増減を自動的に実行します．しかも＋2，－2もします．ポストバイトにより，モー

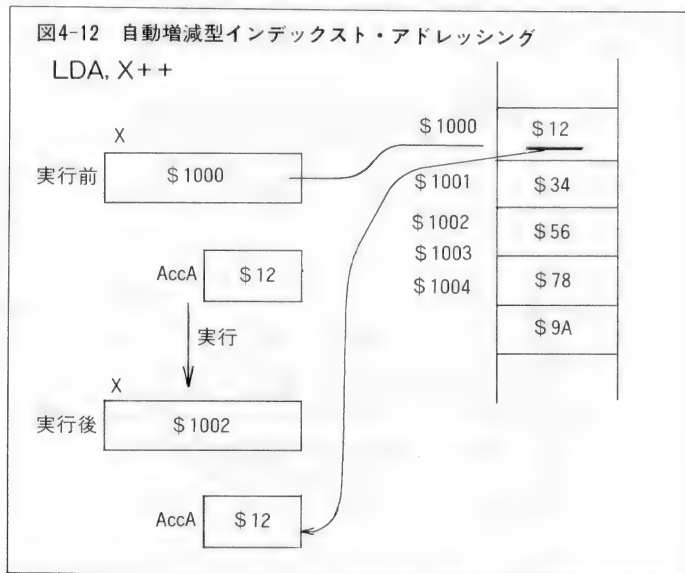
ドは次の４種類があります.

①実行後＋１　　１ＲＲ00000
②実行後＋２　　１ＲＲ00001
③実行前－１　　１ＲＲ00010
④実行前－２　　１ＲＲ00011

ニーモニックにより記法とマシン語はそれぞれ，次のようになります.

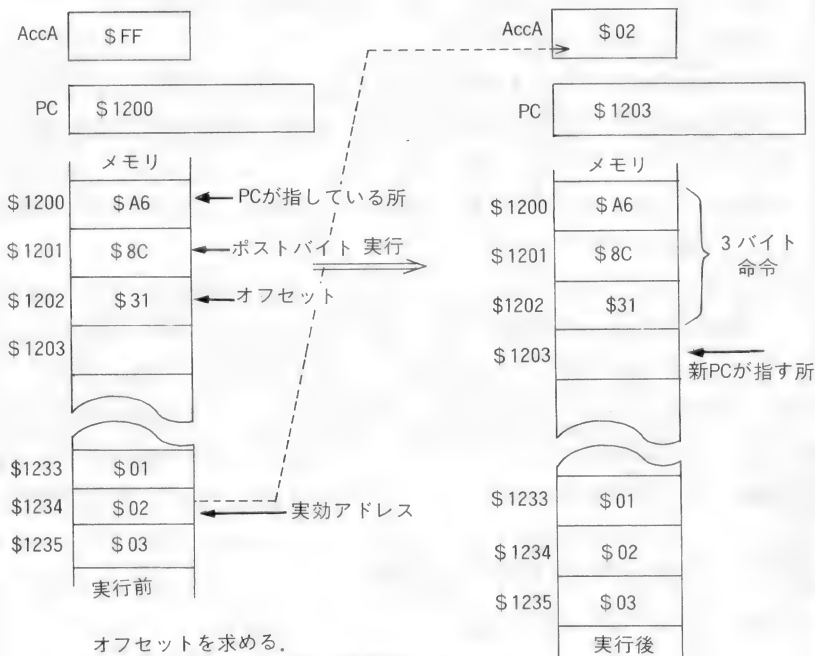
①ＬＤＡ　，Ｘ＋　　＄Ａ６，＄80
②ＬＤＡ　，Ｘ＋＋　＄Ａ６，＄81
③ＬＤＡ　，－Ｘ　　＄Ａ６，＄82
④ＬＤＡ　，－－Ｘ　＄Ａ６，＄83

このアドレッシングの実行例は**図4-12**のようになります.

図4-12　自動増減型インデックスト・アドレッシング

(5)PC相対アドレッシング

ＰＣ（プログラム・カウンタ）をポインタとし，２の補数表現の定数をオフセット値にして加算し，実効アドレスとするモードです.

この形式は，タイトルにあるようにリラティブ（相対）値をとりますので，位置独立なプログラムを書くときに効果を発揮します．実際にこの命令による

図4-13　プログラム・カウンタ相対アドレッシング

データの流れを見てみましょう.

ＬＤＡ　$1234, ＰＣＲ

この場合，命令の実行後のＰＣと目的アドレス＄1234との差（相対値）をオフセットとしたコードを展開します．このニーモニックをマシン語に変換してみましょう．

まず，現在のＰＣの値を＄1200とします．本命令は８ビット・オフセットの場合３バイト命令ですから，この命令が実行された後でＰＣの値は＄1203になっていますので，次式からオフセットを求めます（実効アドレスはあくまで＄1234です）．

$1200 + $3 + x = $1234

x = $1234 − $1203 = $31

これにより，マシン語は次のような3バイト構成になります．

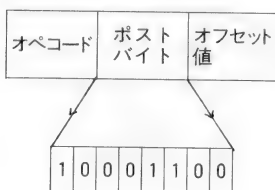
$A6，$8C，$31

2バイトめの$8Cは，8ビット・オフセットのPC相対アドレッシング・モード時のポストバイトです(**表4-1** 参照)．以上の処理の流れは**図4-13**のようになります．

本例はオフセットが8ビット値で表現できる範囲でしたが，16ビット値のオフセットの場合はどうなるでしょうか．

当然ポストバイトの違いによりこのオフセット値の区別をうけることになるため，**図4-14**のような命令構成になります．この命令をハンドアセンブルで書こうとすると，相対アドレスの計算が入ってきて大変です．特に16ビット・オフセットの場合はひじょうに面倒になってくるので，やはり本命令はアセンブラにまかせるのが最良でしょう．アセンブラで処理した例を**図4-15**に示します．この**図4-15**からマシン語は8ビット・オフセット時は3バイトに，16ビット・

図4-14 オフセット値によるポストバイトの違い

a) 8ビット・オフセットの場合

b) 16ビット・オフセットの場合

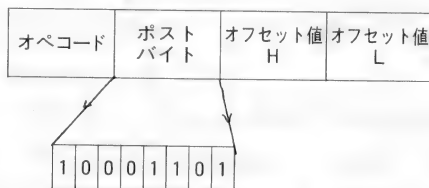



図4-15　PC相対アドレッシングにおけるマシン語

```
PAGE  001 (821201,011346)

00010                                ** PCR モードノ サンプル **
00020  1200                             ORG     $1200
00030  1200 A6    8C 31   8ビット・オ   LDA     $1234,PCR
00040  1203 12            フセット時の  NOP
                          マシン語
00050  1204 A6    8D 0DF8 16ビット・オ  LDA     $2000,PCR
00060  1208 12            フセット時の  NOP
                          マシン語
00070                                   END
```

オフセット時には4バイトに展開されているのがおわかりでしょう．またアドレスもそのバイト数分だけ更新されています．もしハンドアセンブルを行うならば，このようにオフセット値の相違による命令構成数の違いや，相対アドレス値の計算をすべてプログラムのほうで認識して設定する必要があります．

(6)間接アドレッシング

ミニコンとマイコンのアセンブラを比べると，間接アドレッシングに関する命令の豊富さがミニコンのアセンブラの特徴です．逆に，マイコンのMPUは間接アドレッシングの充実よりも，小回りのきいた使いやすい命令の構成に力を注いでいます．しかし，6809は比較的間接アドレッシングが充実したMPUといえるでしょう．

間接アドレッシングについて解説します．一般のアドレッシングでは，オペランドの値をアドレス値としてアドレスの中身を参照しますが，間接アドレッシングで与えられるオペランドの値は，目的とするデータの入っているアドレス（実効アドレス）を示しています．

言い換えると，目的とするアドレスはメモリ中のあるアドレスXに格納し，そのアドレスXをMPUに知らせる形式です．この方式を使用することで，サブルーチンの飛び先をテーブルにして並べ，入力コマンドに相当する処理アドレス先へジャンプする等の場合に大変有効です．ニーモニックによる記法には次のような2つのケースがあります．

①エクステンデット・間接アドレッシング

```
LDA  [$1234]
```

②インデクスト・間接アドレッシング

　　ＬＤＡ　〔2，Ｘ〕

いずれも〔　〕で囲まれたときは間接アドレス（インダイレクト・アドレッシング）を示します．

それぞれエクステンデット・アドレッシング，インデクスト・アドレッシングのときの実効アドレスに従い計算し，その計算結果を目的データの格納されている実効アドレスとする間接指定となります．

2 割り込み動作

IRQ, NMI, RESET, FIRQ SWI1, SWI2, SWI3

割り込み（インタラプト）とは，メインのプログラムが実行されているときに強制的に，別のプログラムの実行をさせる機能です．メインプログラムから見ると，別のプログラムから割り込まれた形になるので，割り込みという表現になるのでしょう．

マイクロコンピュータが制御に大幅に導入されてきているのは，この割り込み機能が充実しているからなのです．逆にいうと，割り込み機能のないコンピュータは使いものになりません．本項では割り込み動作についてわかりやすく説明します．

2・*1*　割り込み機能

68系の割り込み動作は，80系の割り込み動作とは異なります．80系は外部より割り込み命令をデータ・バスに載せるか，ＣＰＵの内部で直接ジャンプアドレスを生成する方式ですが，68系は間接アドレス方式をとっています．

この方式は割り込みが生じると，メモリのある番地に書いておいた2バイトのデータを，割り込み処理プログラムの先頭アドレスとみなして実行します．これらの処理手順のアドレスの一覧表をベクトル・アドレスといいます．割り込みの種類とベクトル・アドレスの関係は**表4-3** のようになります．

＊外部割り込みと内部割り込み

6809の割り込みを大きく分けると，ハードウェアによるもの，ソフトウェアによるものの２種類に分けられます．ＭＰＵの内外という見方から，ハードウェアによる場合を「外部割り込み」，ソフトウェアによるものを「内部割り込み」といいます．

外部割り込みには次の４種類があります．

表4-3　割り込みの種類と割り込みベクトル

優先順位	割り込みの種類	名　　称	割り込みベクトル	
1	RESET　割り込み	Reset Interrupt Request	FFFE	H
			FFFF	L
2	NMI　割り込み	Non Maskable IRQ	FFFC	H
			FFFD	L
3	SWI　割り込み	Software IRQ	FFFA	H
			FFFB	L
4	FIRQ　割り込み	Fast IRQ	FFF6	H
			FFF7	L
5	IRQ　割り込み	IRQ	FFF8	H
			FFF9	L
6	SWI2　割り込み	Software 2 IRQ	FFF4	H
			FFF5	L
7	SWI3　割り込み	Software 3 IRQ	FFF2	H
			FFF3	L

・ＩＲＱ割り込み

・ＮＭＩ割り込み

・ＲＥＳＥＴ割り込み

・ＦＩＲＱ割り込み

また内部割り込みには次の３種類があります．

・ＳＷＩ割り込み

・ＳＷＩ２割り込み

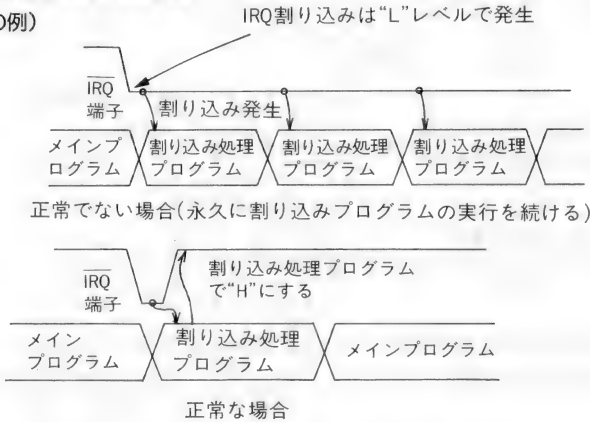
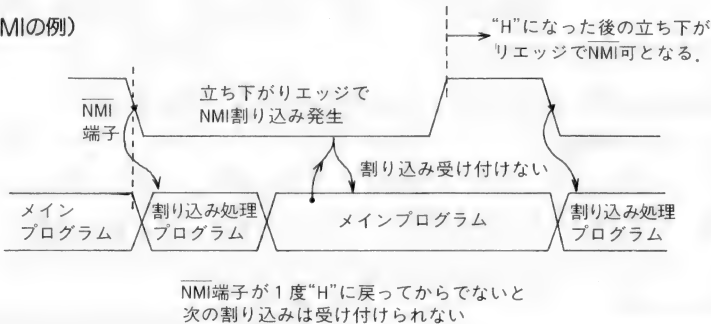
・ＳＷＩ３割り込み

2･2　割り込みの種類

では，各種の割り込み動作を順に説明していきます．それぞれの割り込み動作の違いをしっかりと把握しましょう．

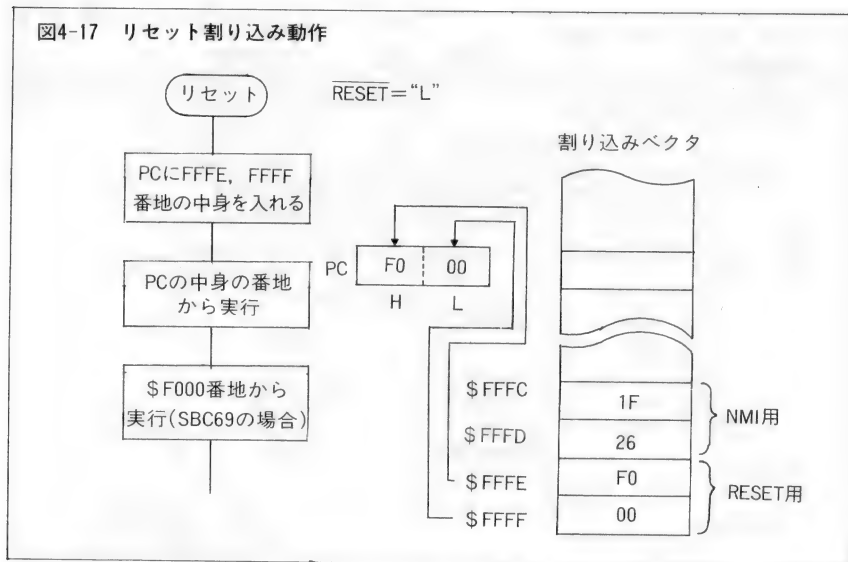
●ＩＲＱ割り込み

ＩＲＱによる割り込みは，ＭＰＵの$\overline{\mathrm{IRQ}}$端子が“Ｌ”になると，すぐに割り込み処理に入らないで，19クロック経過してからそれぞれの処理に入ります．そのためＭＰＵが割り込み動作に入るまで$\overline{\mathrm{IRQ}}$端子を“Ｌ”レベルにしておきます．ただ$\overline{\mathrm{IRQ}}$がいつまでも“Ｌ”になっていると，常に割り込みプログラムの先頭アドレスをフェッチすることを繰り返すので，割り込み処理ルーチンに入ればすぐに割り込み処理プログラムの方で$\overline{\mathrm{IRQ}}$を元の“Ｈ”に戻すよ

図4-16　割り込み信号の性質

うにします.

●NMI割り込み

NMIは割り込み要求が発生すると，MPUはどんな仕事の最中でも今やっている仕事を止めて割り込みプログラムを実行するように作られています（割り込み要求をマスクできない→ノン・マスカブルの意味がここから生じる).

MPUがNMI割り込みを認識できるのは，$\overline{\text{NMI}}$端子が立ち下がりエッジで“L”になるからです．一度，NMI割り込み処理を実行してしまうと$\overline{\text{NMI}}$が一度“H”になった後でないと，再び割り込み処理を実行することはできません．**図4-16**に割り込み処理の性質を示します.

ＩＲＱ割り込みとＮＭＩ割り込みの違いを理解しておきましょう．

●ＲＥＳＥＴ割り込み

リセット動作は割り込みの一種です．ＲＥＳＥＴが“Ｌ”レベルになるとＦＦＦＥ，ＦＦＦＦのそれぞれがアドレスに出力され，ＦＦＦＥ番地とＦＦＦＦ番地に（一般にはＲＯＭを割り当てる）システムのリスタート・プログラムの先頭番地を割り当てておくことで，システムはリスタート動作を実行します．この動作は6800とは異なり，パワーオン時のみクロックＱ，Ｅが安定するまでＲＥＳＥＴを“Ｌ”にしておく必要があります．一度立ち上がってからのリセット動作は１クロック以上を“Ｌ”にするだけで大丈夫で，6800より大部高速になっています．**図4-17**に，本機におけるＲＥＳＥＴ割り込み動作の流れを示します．

この図でわかるように，ＳＢＣ69は電源ＯＮ直後やＲＥＳＥＴキーを押した後は必ず＄Ｆ200 番地から実行します．そのためモニタＲＯＭを配置しておけば，モニタＲＯＭの内容のプログラムを実行します．

●ＦＩＲＱ割り込み

ＩＲＱ割り込みが19クロックもかかってから処理されるのは，全レジスタをスタックに退避しているためです．しかし，この退避はすべてのレジスタを退

避させる必要がないときは大きな時間の無駄となってしまいます．そこで，CCとPCだけを退避して割り込みプログラムへ飛ぶ高速IRQ割り込みが用意されています．

●SWI割り込み

“Software Interrupt”の名称が示すように，命令で割り込み処理に入れる機能です．

MPUは，このSWI命令（マシン語で3F）に出会うと，全部のレジスタをセーブしてFFFA番地とFFFB番地の内容を先頭とする割り込みプログラムに飛びます．とくにモニタ動作における1ステップ動作時のレジスタ表示等のデバッグ機能に効果を発揮します．6809には，このほかにSWI2，SWI3の割り込み命令があります．

2・3　割り込みからの復帰

割り込み要求が発生すると，MPUはレジスタに格納されているデータの全部または一部をスタックにセーブしてから割り込み処理を実行しますが，割り込み処理が終わってもそのままメインプログラムには戻りません．

割り込み処理が終了したら，スタックに保存（＝セーブ）していた各レジスタの中身を元に戻してから，メインに戻る必要があります(**図4-18**)．その処理を行うのがRTI命令になります．

図4-18　割り込み時のレジスタの保持

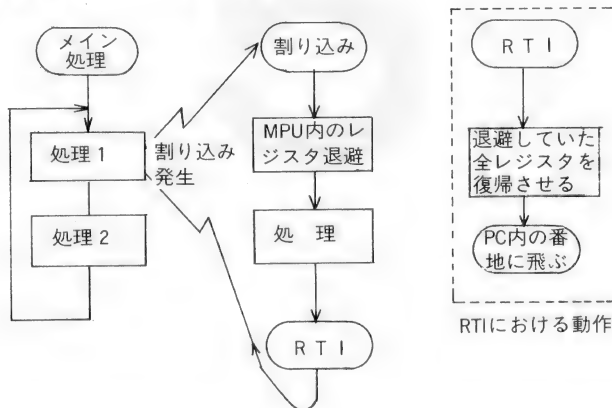
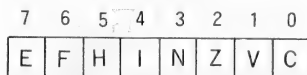



図4-19　CCレジスタ

レジスタを戻すときには，全部を戻すのか一部を戻すのかを知る目印が必要です．この目印すなわちフラグに相当するのがＣＣのＥフラグです（**図4-19**）．

ＣＣの７ビットめが“１”であれば，全レジスタが退避されていますので，ＲＴＩ命令のときに全レジスタを復帰させます．また“０”であれば，ＰＣとＣＣだけを復帰させます．このＲＴＩの処理は，ＭＰＵ側が自動的に実行します．

Chapter Five

回　路　設　計

5

本章では，第２章までに学んだ基本事項をベースにして，具体的なワンボード・マイクロコンピュータの設計を行っていきます．

市販のパーソナルコンピュータを購入した学習形態では絶対に経験できない分野がこれから学習していく部分です．自分で回路を組み，理解していくわけですから，全体の大きな回路も，実は小さな基本回路の集まりであるということが理解できることでしょう．

一度SBC69を製作した後は，自分なりに設計し直したり，アレンジしたり拡張して，さらに高機能のコンピュータへ発展させていくことも夢ではありません．

たくさんの事項が出てきますが，あせらないで一つ一つじっくりと，自分が設計するつもりで進んでいってください．

1 設計の基本方針

使用形態，支援ソフト，アドレスマップ

1・*1*　SBC69の基本仕様

まず，製作するコンピュータに名前をつけましょう．“シングルボード・コンピュータ6809”の頭文字をとって「ＳＢＣ69」と名付けましょう．

さて，名前が決まったら，次にこのコンピュータが備える仕様を決めていきます．製作するマイコンの主な仕様は次のようになります．

・ＭＰＵ	68Ａ09　８ビット　１メガヘルツクロック		
・メモリ	ＲＡＭ領域	6264スタティックＲＡＭ使用　８Ｋバイト	
	ＲＯＭ領域	2732×２	①ブートストラップ用　４Ｋバイト
			②モニタプログラム記憶用　４Ｋバイト・I/O
	ＰＩＡ（6821Ａ）		16進キーボード・ドライブ用
	ＡＣＩＡ（6850）		ホストＣＰＵからのデータ転送用
・直列信号	転送スピード		300　ボー～9600ボーまで選択
・動作状況チェック用モニタ機構			ＬＥＤにより指示表示
・データ出力表示			７セグメントＬＥＤにより表示

1・*2*　設計思想

マイコンを製作するときのもっとも大事な決定事項はメモリマップ（メモリ配置）です．いったんメモリマップを決めてしまえば，ソフトの関係上あとから変更することは容易でありません．そこでＳＢＣ69の設計では，設計思想を次のように考えました．

① ホストＭＰＵ（今回は富士通ＦＭ-8，ＦＭ-7にした）のメモリ配置と同一にする．こうすることで，ＦＭ-7で走っていたI/O 用機器がすべて接続可能

になるし，開発においてもホストMPUのアセンブラがフルに活用できる
② アドレスはできるだけフルデコードして，イメージをなるべく作らない
③ ほかのMPUマシンもホストにできるように，できるだけ拡張用端子に特殊な信号線を引き出さない

以上の思想をベースにして設計を進めていきましたので，ＳＢＣ69は「山椒は小粒でもピリッと辛い」小回りがきくマシンに仕上がりました．単独で使用する狭い範囲の自作ワンボード・マイコンではなく，ホストマシンのエミュレータ用にも使えるなど，幅広い応用に使うことができます．

1•3　使用形態

このワンボード・マイコンを使用するときの形態について考えてみましょう．使用形態には一般的に次の２つの方法があります．

①16進キーで直接メモリ入力する

②16進キーでホストコンピュータを使って間接メモリ入力する

①は，今までのワンボード・マイコンでよく行われてきた16進キーから直接メモリにマシン語で入力してから実行する形態です．

この形態は，マシン語を直接キーからメモリに打ち込むわけですが，実際やってみると１Ｋバイトのデータを入力するのでさえも，相当な負担です．私も昔，ワンボード・マイコンでプログラムを開発していた頃，４Ｋバイトを入力するのに大変苦労した苦い思い出が数多くあります．

この経験から，今回は①の使用形態を用いず，②の使用形態で行います．

図5-1はステッピングモータを回転させる実験を行う例です．この実験例を使って②の動作内容を説明することにしましょう．

＊使用形態の動作

16進キーは，ホストマシンからのデータの転送開始の指令と，転送されてきたプログラムに実行する命令を与えるときにのみ使います．ユーザはホストマシン上においてソース・プログラム（アセンブリ言語で書かれたプログラム）を作り，オブジェクト・プログラム（マシン語で書かれたプログラム）に直します．そしてこのプログラムをＳＢＣ69に転送し，ＳＢＣ69のメモリに格納してから16進キーでＧＯ命令を与え，ステッピングモータが回転します．

②の動作形態の特徴を**表5-1**に示します．

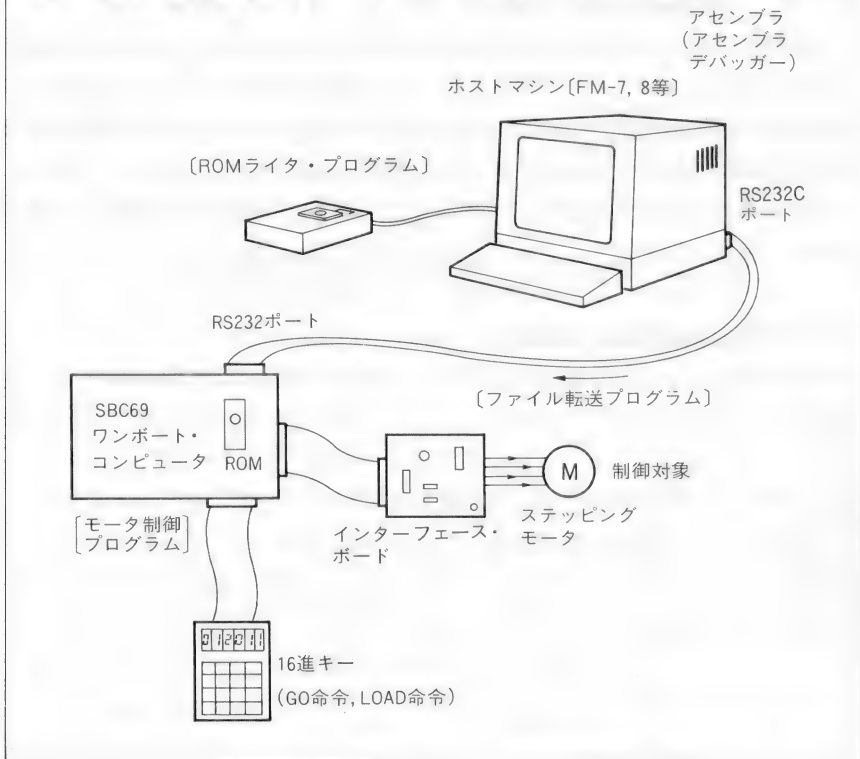
図5-1　使用形態と支援ソフト

ホストマシンは，直列転送用ポートを備えていてMPUに6809を使用しているコンピュータならどれでも構いませんが，ここではFM-7，FM-8クラスのコンピュータを使用しました.

表5-1　使用形態の特徴

特　長	欠　点
・ホストマシンでアセンブラ等を使いながらプログラムの開発が可能 ・ホストマシンをデバッガとして代用しデバッグ作業ができる. ・プログラムの開発が非常に短縮できる ・ホストマシンの開発用資源がすべて使える	・直接マシンを動かし、プログラムを実行させたという満足感が少ない ・ホストマシンの方に開発用のツールが充実していることが必要 ・マシン語のみでワンボード・コンピューを操作するよりも広範囲の知識が要求される

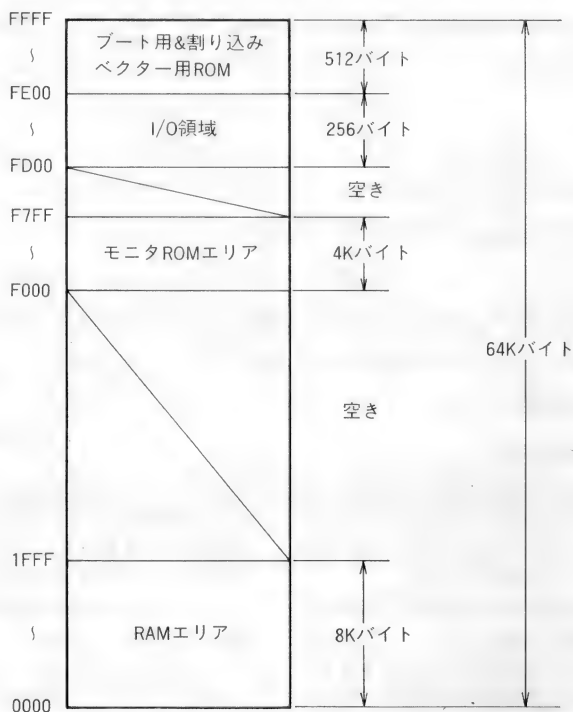
1・4　SBC69のメモリマップ

ＳＢＣ69のメモリ容量はどのくらいになるのでしょうか．ＳＢＣ69の心臓であるＭＣ6809は８ビットですから，全メモリ空間は64Ｋバイトになります．この限られたメモリ空間で，ＲＯＭ，ＲＡＭ，I/O用ＬＳＩのメモリの配置を考えなければなりません．メモリマップ（配置図）を**図5-2**に示します．

同図を見ればわかるように，メモリマップには空き領域があります．この領域はボード上のアドレスデコーダを変更すれば自由に使えます．ワンボード・マイコンとして使用するには，本機のメモリ容量でじゅうぶんです．

＊ I/O 領域は主記憶エリアの中

80系のコンピュータを使用してきた人に奇妙に映るのがI/O領域です．実

図5-2　メモリマップ

は，68系では80系のように主記憶エリアとは別のI/O 専用ポートを持っていません．入出力機器もメモリの一部として主記憶エリアの中に確保する方式を採用しています．I/O 領域は，MPUと入出力装置との間データのやりとりのために割り当てられているメモリです．

68系の方式はメモリを消費しますが，メモリに対する強力な種々の命令がI/O 領域でも使えますので，80系のＩＮ，ＯＵＴ命令だけの入出力処理よりもはるかに高機能な入出力用プログラムを書くことができます．

1・5　I/O領域のアドレスマップ

コンピュータの入出力装置のことを一般にI/O（Input/Output）と呼びます．I/O は人間とコンピュータのコミュニケーションにとって必要なものです．たとえMPUがプログラムを実行しても，I/O がないとディスプレイに表示することも，プリンタに打ち出すこともできません．

メモリ64Kバイトの中にあるアドレス空間には，ＭＰＵと入出力装置とのデータのやりとりをする空間が設けられています．その空間がI/O 領域です．I/O（入出力）領域は図5-3（次ページ）のようになります．割り付けはＦＭ-7，ＦＭ-8のコンピュータと同一になるようにしています．図中の＄ＦＤ00から＄ＦＤＦＦまでがＳＢＣ69のI/O 専用のメモリ領域です．

I/O 領域を占めている素子には３つの素子があります．ＰＩＡ，ＡＣＩＡ，ＰＴＭの素子です．

これらのI/O 素子の働きを以下に簡単に説明します．

＊ＰＩＡ（Peripheral Interface Adapter）

ＰＩＡの#１は，ＳＢＣ69のシステム用ＲＯＭの中のモニタによって使用されます．ユーザにはＰＩＡの#２の方を開放していますので，ＰＩＡの#２にＬＥＤやモータ等を接続して制御します．

＊ＡＣＩＡ（Asynchronous Communications Interface Adapter）

ＡＣＩＡは，ＳＢＣ69本体のデータを取り込む大事な機能を受け持ちます．ホストコンピュータの直列信号出力ポートと連結して使用します．

＊ＰＴＭ（Programmable Timer Module ）

ＳＢＣ69では拡張I/O ボードを後で準備しますが，ＰＴＭはそのときに使用し，タイマ割り込みに関する部分の機能を受け持ちます．

図5-3 I/Oメモリマップ

SBC69の全体回路

MPU, バッファ, リセット アドレスデコーダ, メモリ

＊ＳＢＣ69の全回路

図5-4にＳＢＣ69ワンボード・コンピュータの全回路を示します．ここでは，この回路図をもとにして各部の動作を説明していきます．

2•*1*　MPU回路

＊水晶振動子は４ＭＨz

ＭＰＵは第２章で述べたようにＭＣ6809Ｅではなく，6809です．6809は内部に発振回路を持っているので，6809のＸＴＡＬ端子とＥＸＴＡＬ端子に水晶振動子を接続するだけで発振します．6800と比べると，ずいぶん楽になりました．ただ内部で１/４に分周されるので，クロック周波数の４倍の振動子を接続しなければなりません．本機では，システムクロックを１ＭＨzにしたため，４ＭＨzの水晶振動子を接続します．

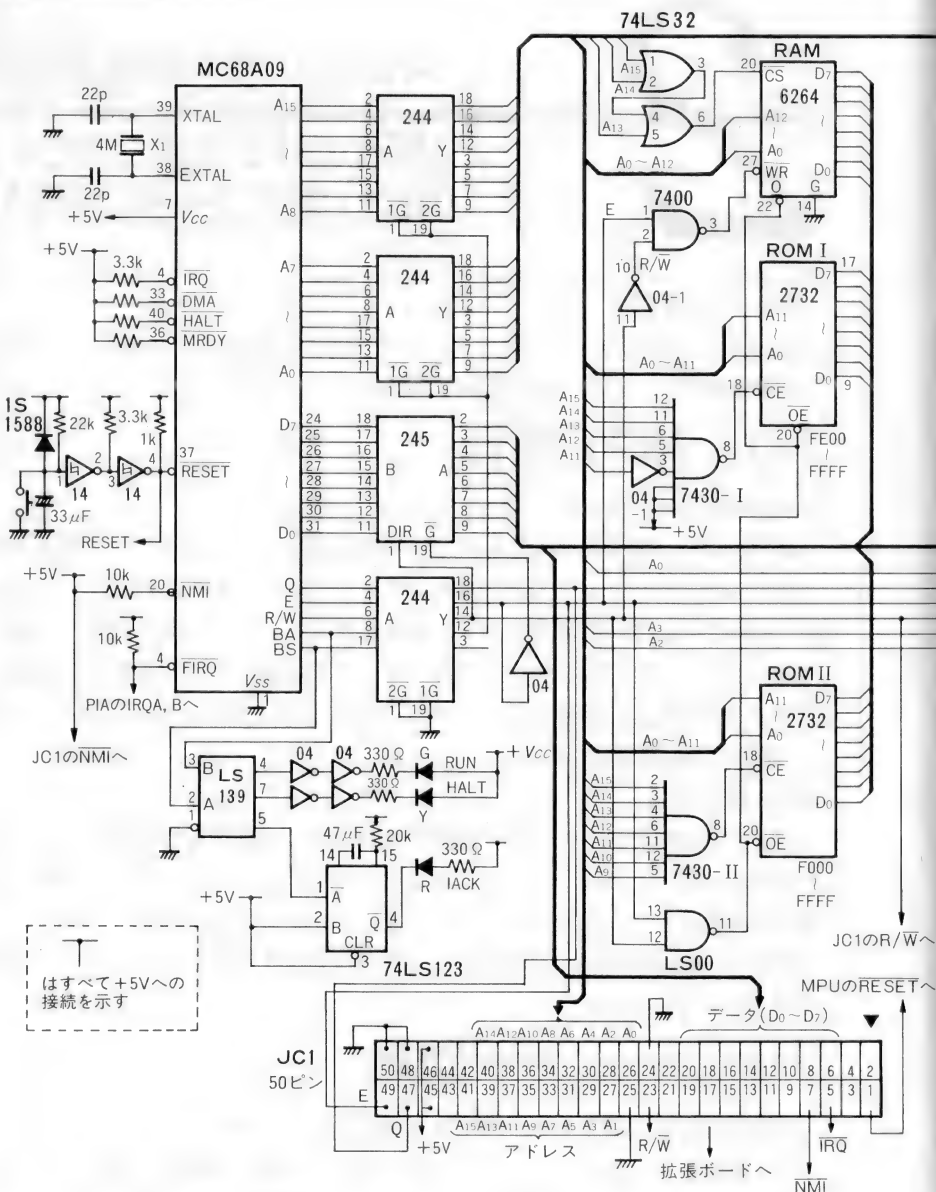
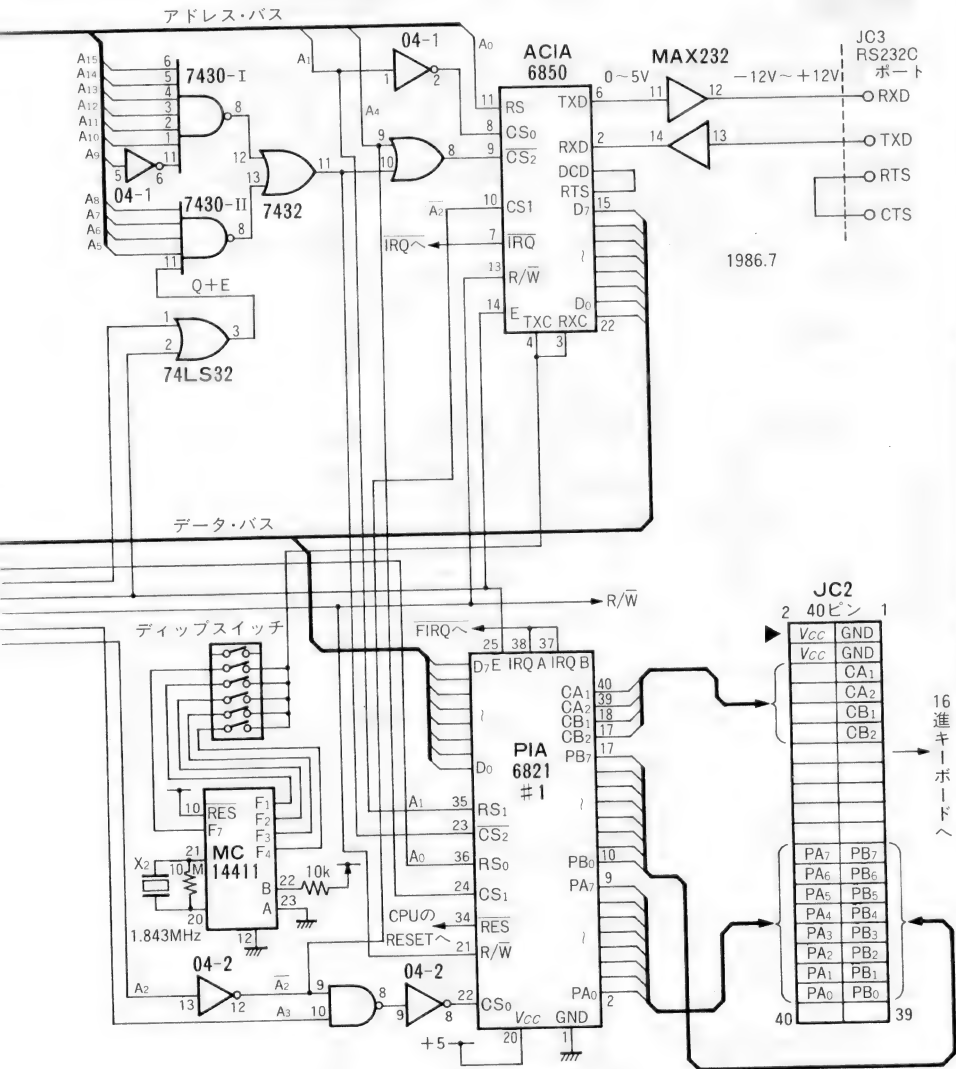
＊ＭＰＵチップ

ＭＰＵチップの端子は，最短距離で結線しないと不安定な動作をします．22ＰＦのコンデンサはセラミック・コンデンサでじゅうぶんです．また使わない端子は開放にしておかないで，すべてプルアップしておきます．

図5-5に発振周波数とＣの関係を，**図5-6**に要求されるシステムクロックＥ，Ｑの波形を示します．

2•*2*　バッファ回路

本機はＭＰＵと周辺のチップ間は直結しないで，間にすべてバッファを入れてから拡張しています．バッファは弱いデジタル信号の電流を増強するために使われます．バッファ回路をブロック図にして**図5-7**に示します．また，バス・

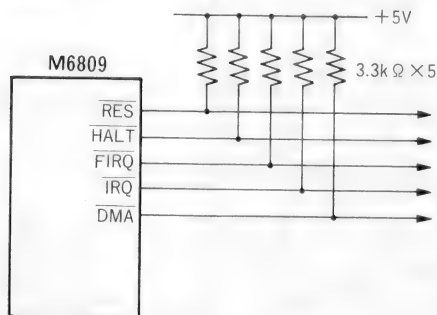
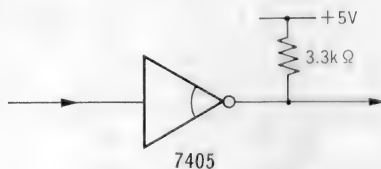
図5-4　SBC69の全体回路

アドレス・バス
データ・バス
7430-Ⅰ
7430-Ⅱ
7432
74LS32
04-1
04-2
Q+E
ACIA
6850
MAX232
0～5V
－12V～＋12V
JC3
RS232C
ポート
RXD
TXD
RTS
CTS
1986.7
IRQへ
R/W
FIRQへ
ディップスイッチ
MC
14411
1.843MHz
10M
10k
PIA
6821
♯1
CPUの
RESETへ
+5
JC2
40ピン
16
進
キ
ー
ボ
ー
ド
へ
・TTL ICは全てLSタイプ
・論理は正論理で示している

column
コ・ラ・ム

プルアップ抵抗

TTL ICなどで出力側に多数のIC入力端子を接続したり，負荷を接続したりすると，出力レベルが＋５Ｖの値からどんどん減少していき，ついには“１”か“０”かの判定が困難になります．このようなとき外部から，その出力端子に電源を供給して出力をＨレベルに強制的に上昇させます。この電源供給用に使用する抵抗をプルアップ抵抗といいます．

図は、プルアップ抵抗の使用例です．なお，この逆の働きもさせることができますが，この場合にはプルダウンするといいます．

(a) 一般的例

(b) オープンコレクタICの場合

(c) スイッチの場合

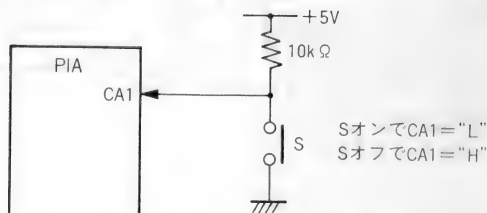



図5-5　発振周波数とCの関係

Y1	Cin	Cout
8 MHz	18 pF	18 pF
6 HMz	20 pF	20 pF
4 MHz	24 pF	24 pF

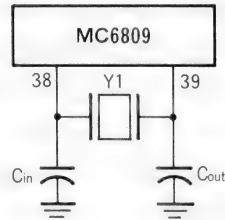



図5-6　必要なシステムクロックの波形

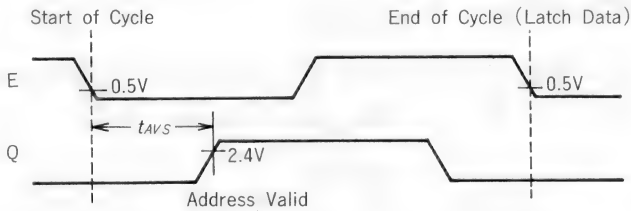
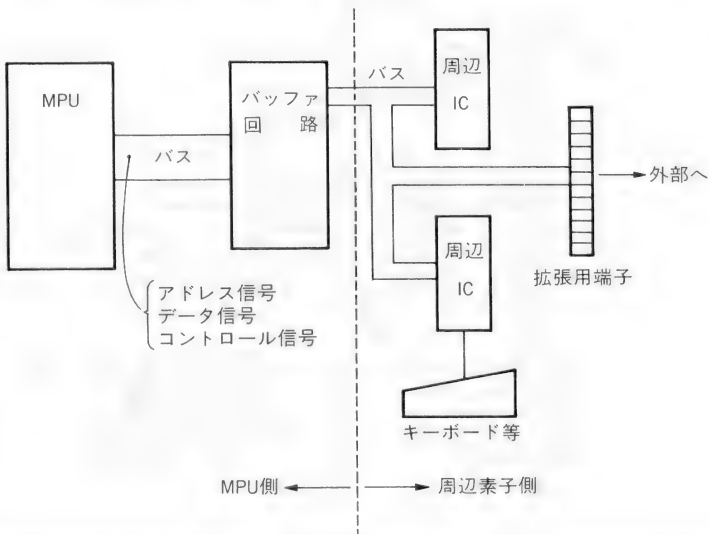
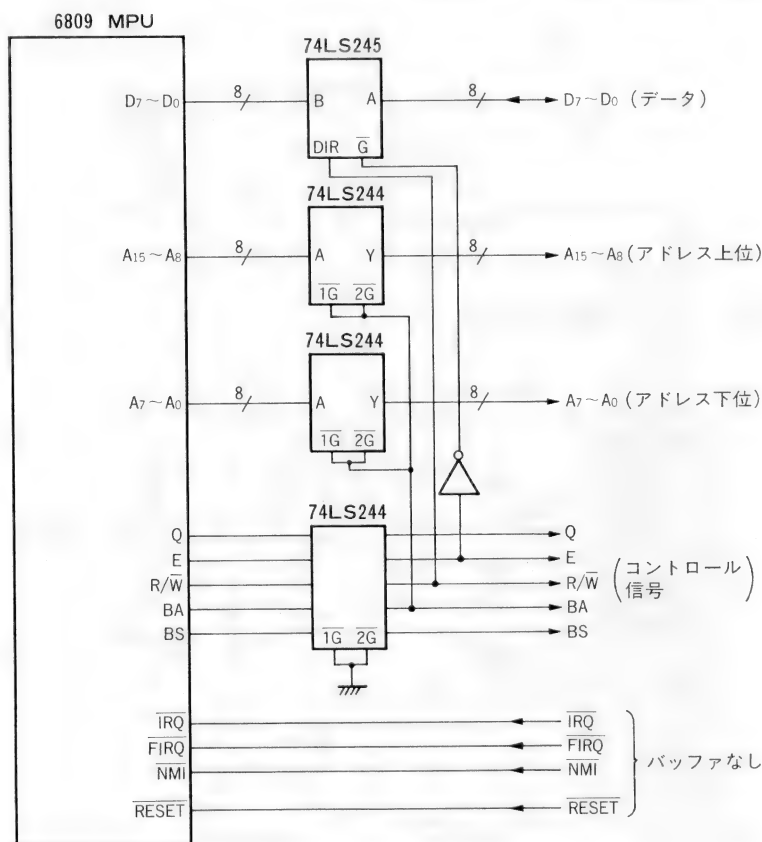



図5-7　バッファ回路のブロック図

バッファの全体回路は**図5-8**のようになります．

＊バス・バッファ

バスには，アドレスがやりとりされるアドレス・バス，データがやりとりされるデータ・バス，それに信号がやりとりされるコントロール・バスの３種類があります．このうちデータ・バスはＩＮ，ＯＵＴの双方向性がありますが，アドレス・バスはアドレス値をＭＰＵから周辺に向かうＯＵＴ信号だけの一方向バスです．

図5-8　バッファ回路の全体図

コントロール・バス上の信号はMPUにとってINになる信号とOUTになる信号の両方がありますが，本機では回路を簡単にするために，MPUにとってOUTになる側の信号だけをバッファリングすることにしました．

では，MPUとバッファ回路をどのように制御したらよいか，各々の場合について考えてみましょう．

●アドレス・バス・バッファ回路

MPUのチップ端子にBA，BSという端子があることを思い出してください（2章P48）．BAが“1”のときはMPUが停止状態であり，BAが“0”のときはMPUが走行状態であることを示します．またアドレス値とBAの値は同時に立ち上がりますから，この信号によりバッファの切り換えができるわけです．

バッファ回路用として使用できるICは74LS244です．**図5-9**に74LS244の信号伝送方向の切り換え状況を示します．またBA信号によるドライブ回路は**図5-10**のようになります．

74LS244は1個の中に8素子が組み込まれているので，アドレス・バッファとしては2個必要です．74LS244の内部結線を**図5-11**に示します．

図5-9　アドレス・バッファ回路

G端子	信号伝送方向
L	A → Y（導通）
H	ハイインピーダンス(断状態)

図5-10　BA信号によるドライブ回路

MC6809

A15 A14 A13 A12 A11 A10 A9 A8

A7 A6 A5 A4 A3 A2 A1 A0

BA

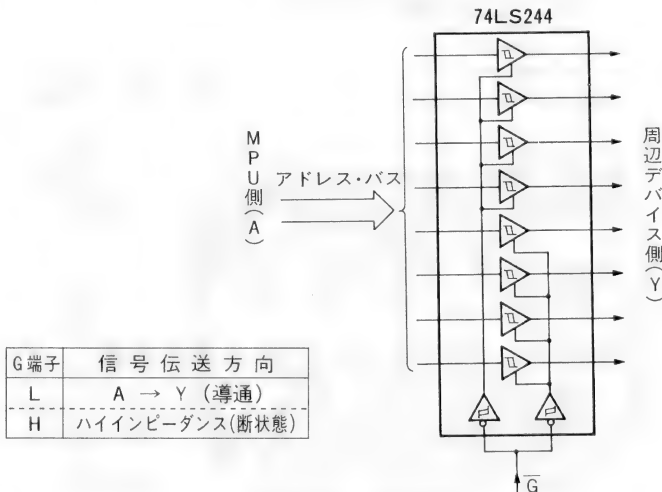
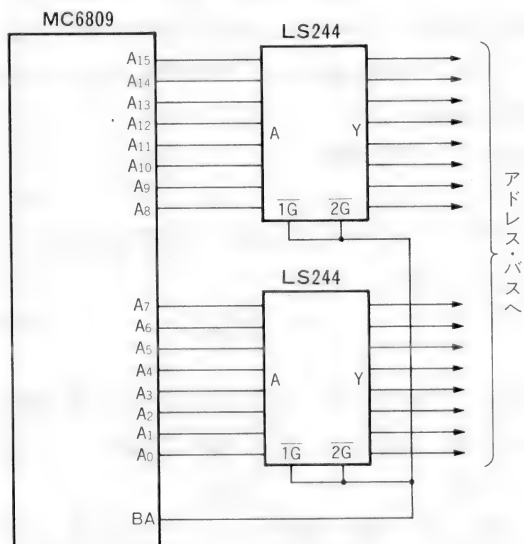
LS244

A　Y

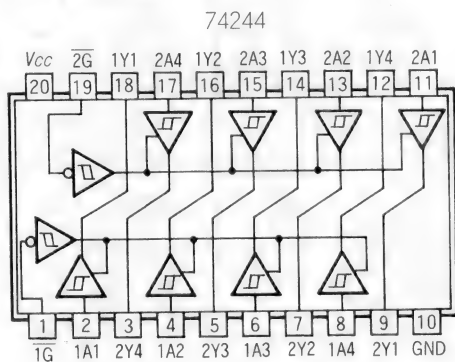
1G　2G

LS244

A　Y

1G　2G

アドレス・バスへ

BA	MPU状態	信号の向き
H	停　止	断
L	走行(通常)	A → Y

図5-11　バッファ74LS244

●データ・バス・バッファ回路

アドレス・バスは信号がMPUから周辺素子へ流れる単一方向なので回路は簡単でしたが，データ・バスの信号はMPUに入ってくる入力信号，MPUから出ていく出力信号の2つの方向を持っているので，その制御回路は少し複雑になります．そのためデータ・バスの制御では，次の2つの事柄が重要な要素になります．①いつバッファを開いて信号を流すか，②入力，出力の切り換えはどこで行うかの2つです．

①の条件を理解するために，**図5-12**，**図5-13**を見てください．この2つの図

図5-12　6809のリード時のタイミング

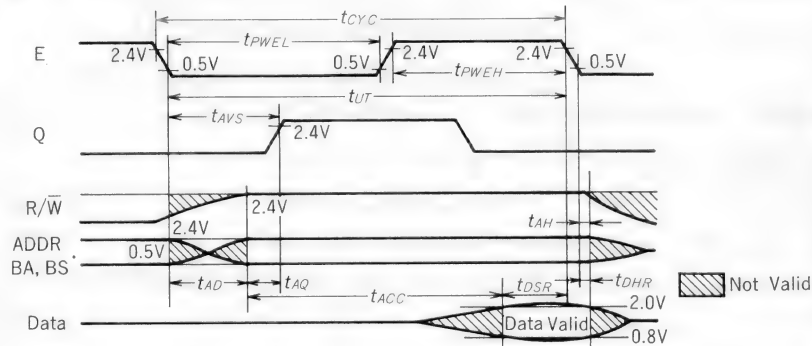
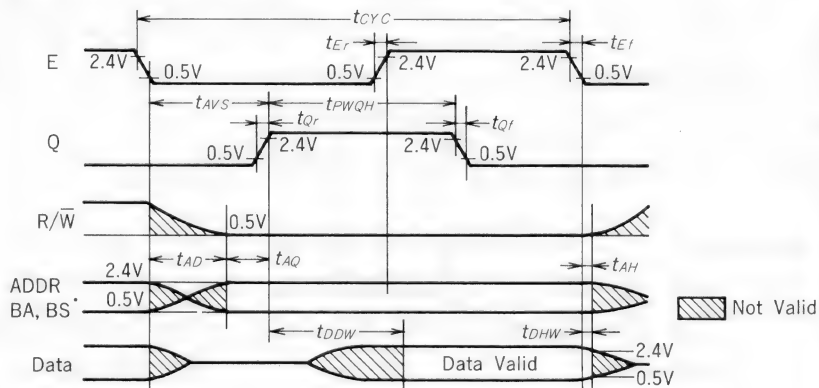



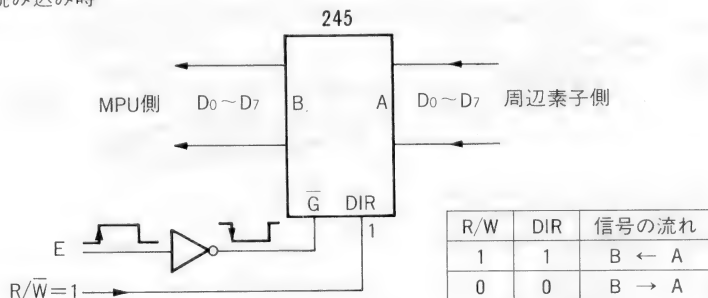
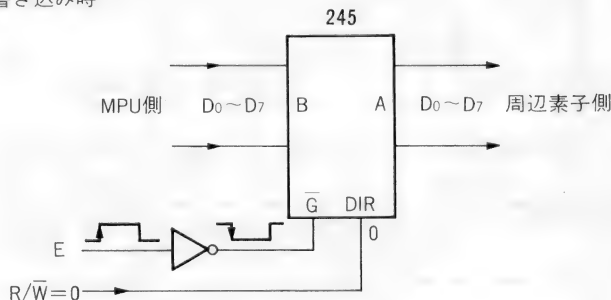
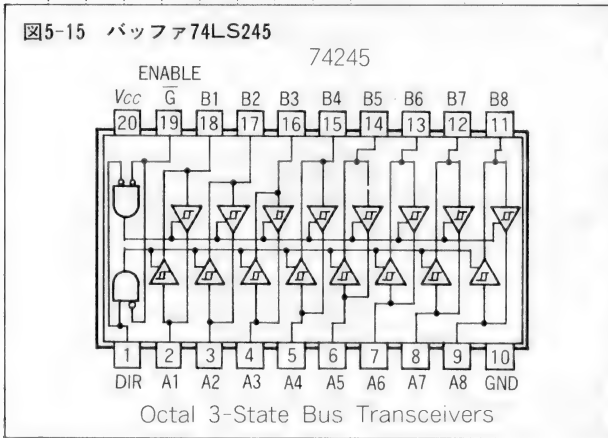
図5-13　6809のライト時のタイミング

は6809におけるデータのリード（入力）とライト（出力）のときのタイミング図です．

このタイミング図からわかることは，すべての信号がクロックEとQのタイミングに同期して動き，E信号の“H”の範囲でアドレスが確定することですこの区間バッファを開くために，E信号を反転して74ＬＳ245のG端子に接続すれば，①の条件は満足します．

タイミング図のR/$\overline{W}$を見てください．データをMPUが入力（読み込み）するときはR/$\overline{W}$端子が“H”になり，MPUがデータを出力（書き込み）するときはR/$\overline{W}$が“L”になっています．したがって②の条件を満足するためには，R/$\overline{W}$端子を74ＬＳ245のＤＩＲ端子に接続すれば74ＬＳ245の信号の流れる方向を反転できます．ここの部分の説明を**図5-14**に示します．

図5-14　74LS245の方向切り換え

R/W	DIR	信号の流れ
1	1	B ← A
0	0	B → A

図5-15　バッファ74LS245

また**図5-15**は74ＬＳ245の内部配置の様子です．

●コントロール・バス・バッファ回路

コントロール・バスにはＭＰＵから周辺素子に向かう信号と，周辺素子からＭＰＵに向かう信号の２種類があります．本来なら，すべてコントロール・バスもバッファを通しますが，両方向用のバッファを用意すると基板のスペースあるいは回路的にみても複雑な配線になります．そこで今回は必要最小限の信号のみバッファを通すことに決め，ここではＥ，Ｑ，$R/\overline{W}$，ＢＡ，ＢＳをバッファすることだけにとどめました．

2･3　リセット回路

マイクロプロセッサではリセットも一種の割り込みになっています．この信号がＯＮになると，必ずＭＰＵはメモリ上のある特定番地に飛ぶように作られています．したがって，システムを初期化して立ち上げるルーチンをこの番地に書いておけば，リセットが働くとシステムは必ず初期状態に戻り，再起動します．

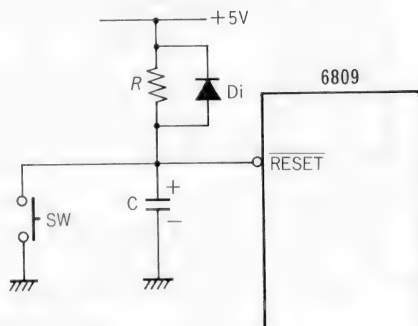
このように，リセット回路はシステムを確実に立ち上げ，また暴走状態になったときには確実に安定な初期状態を取り戻す大切な役目を持っています．本機では**図5-16**に示すようなＣの充放電による方式にしました．

＊Ｃの充放電方式とは

スイッチを押すことで$\overline{RESET}$は“Ｌ”になり，スイッチを離すことでＣ

が徐々に＋５Ｖ側に充電されていき，やがて＋４Ｖ以上では定常状態に入っていきます．リセットの場合“H”レベルは４Ｖ以上と規定されているので，実際は全体回路図で示しているように１ＫΩでプルアップしておくのが確実です．

図5-16のＤｉはＣの放電を早めるために必要な端子です．Ｄｉがなければ，一度電源を切った場合再び電源を入れ直すまでの時間が必要です．

図5-16　パワー・オン・リセット回路

2・4　アドレスデコーダ回路

アドレスデコーダ回路は，ＭＰＵに接続する１つ１つの入出力（I/O）ＬＳＩやメモリ等の素子に番地（アドレス）を与えて，ＭＰＵがそれらの素子をアドレスで呼べるようにします．とても大事な部分ですから，アドレスの割り付けが１番地でもずれてしまえば絶対に動作しません．入出力素子もすべてメモリの一部とみる68系ＭＰＵではとくに重要な回路です．このへんの関係を図5-17に示します．

(1)　アドレスデコードと素子選択の関係

では，アドレスを与えることによってＭＰＵはどのように素子を選ぶのでしょうか．

I/O 素子には必ずチップ・セレクタという端子があります．図5-18のＰＩＡの場合にはCS_0，CS_1，$\overline{CS_2}$というように３種類があります．端子CS_0は１に，CS_1も１に，$\overline{CS_2}$は０にそれぞれ接続すると，ＰＩＡがＭＰＵに選んでもらえる条件が１つできます．

このように，選択素子のチップ・セレクトＣＳ端子の条件にあった値（ＣＳ

図5-17　I/O素子もメモリと同格

メモリ領域(0000H～FFFFH)

番地	メモリ
1234番地	PIA
1235番地 ～ 1356番地	メモリ
1357番地	ACIA
1358番地 ～ 1623番地	メモリ
1624番地	PTM
1625番地	メモリ
	メモリ

I/O素子

PIA ※1234H番地

ACIA ※1357番地

PTM ※1624番地

68系においてはI/O素子もメモリも同格の扱いを受ける

(※番地の値はSBC69の割り付け番地ではなく，仮の値です)

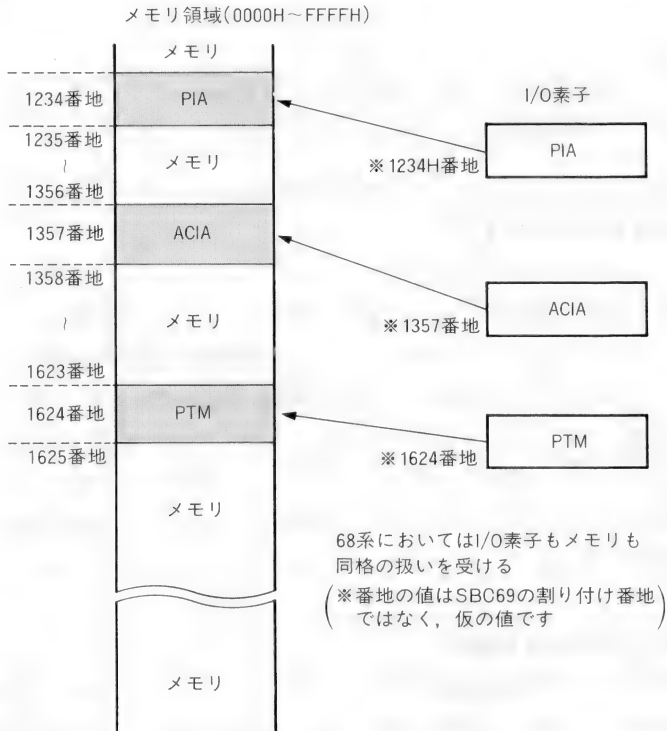



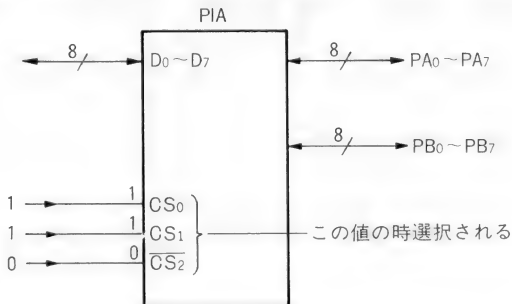
図5-18　チップ・セレクトによる素子選択

素子名	IC名	割り付けアドレス
RAM	6264	0000 ～ 1FFF
ROM I	2732	F000 ～ F7FF
ROM II	2732	FE00 ～ FFFF
PIA	6821	FDF8 ～ FDFB
ACIA	6850	FDE0 ～ FDE1

表5-2
素子と割り付けアドレス値の関係

は1，$\overline{\mathrm{CS}}$は0）を与えれば，素子が選択されます．

(2) 素子と割り付けアドレス

表5-2に，実際の素子と割り付けられるアドレスを示します．この表にある範囲のアドレスが指定されたときに素子が選択されるように回路を工夫します．**図5-19** (1) (2) (3)にアドレス値によるデコーダ回路の基本図を示します．

ＡＣＩＡのときにA_3の線はどこにも接続されていないため，A_3は“１”でも“０”でもよいことになります．したがってＡＣＩＡを選択できるアドレスはＦＤＥ０，ＦＤＥ１番地だけでなくＦＤＥ８，ＦＤＥ９でも構いません．

このように，アドレス・デコードの手抜きのために生じる規定外のアドレスをイメージ・アドレスといいます．今回はＦＤＥ８，ＦＤＥ９がイメージ・アドレスです．

(3) デコード回路の全体構成

具体的なデコード回路は前項で示しましたが，みなさんの中にはなぜこんな複雑な回路にしたのかと疑問を持つ人がいると思います．すべてFM-7，FM-8の汎用コンピュータと同じアドレス割り付けにしたためにこのようになりました．組み込みワンボード・マイコンのようにアドレス・デコード域を自由に設定するときは74139等を使って能率的で部品類が少ない回路構成ができます．

図5-19 素子とデコーダ回路

1. RAM

アドレス＼ビット値	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
～		～				～				～				～		
1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

2. ROM I（モニタROM）

アドレス＼ビット値	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
F000	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
～		～				～				～				～		
F7FF	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1

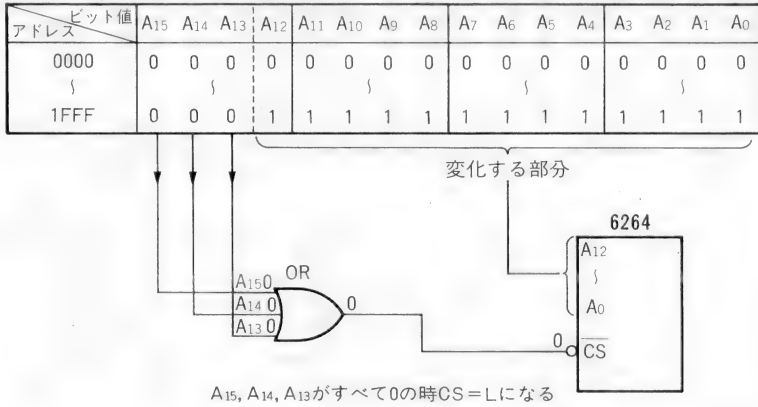
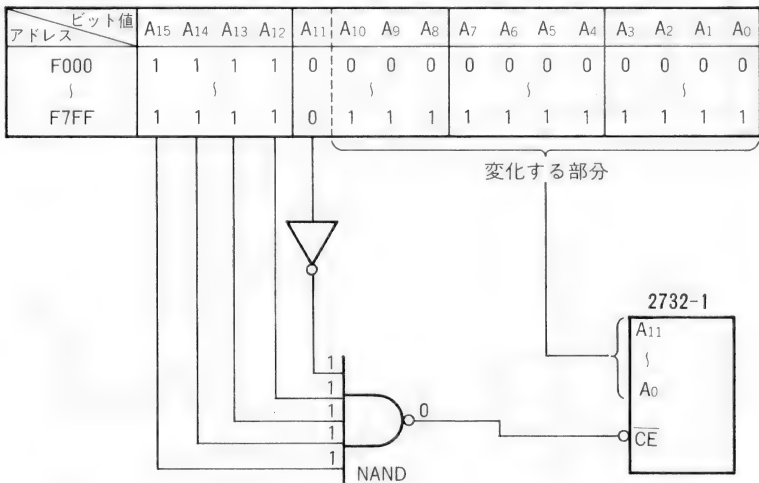



図5-19のつづき

3. ROM II（ブートストラップ用ROM）

アドレス＼ビット値	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
FE00	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
〜		〜					〜			〜				〜		
FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

4. PIA（システム用）

アドレス＼ビット値	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
FDF8	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0
FDF9	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	1
FDFA	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	0
FDFB	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1

図5-19のつづき

5. ACIA

ビット値 / アドレス	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
FDE0	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0
FDE1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	1

接続しない

ACIA

RS

CS_0

CS_1

$\overline{CS_2}$

NAND

NAND

A_3を接続しないためA_3は1, 0のどちらでもACIAは選択されることになる
………………イメージアドレスの発生

2・5 メモリ回路

本機に使用されているメモリ素子はRAMとROMに分けられます．２つのメモリ回路は次のような構成になっています．

●RAM6264

6264はスタティックRAMです．64Kビットの記憶容量と，８Kバイトの記憶エリアがあります．ダイナミックRAMのようなリフレッシュ操作はありませんので，非常に使いやすい素子です．またリセットをかけてもデータは保存されるので大変便利です．**図5-20**に6264のピン接続を示します．

図5-20 RAM6264のピン配置

＊RAM6264のアクセス法

6264のアクセス回路は**図5-21**のようになります．6264のチップを選択し，データの入出力動作をさせる端子は$\overline{CS}$と$\overline{WR}$，$\overline{OE}$の３つです（**図5-22**）．

＊アクセスタイムとタイミング

スタティックRAMのようなメモリが，データのやりとりをするときに大切なことはアクセスタイムです．

図5-12，図5-13に示したリード，ライトのタイミングでは，I/Oと同じようにメモリもデータの読み書きをします．6809は80系と異なりクロックに同期した同期バス方式ですから，MPUがメモリとやりとりするときのタイミングは決まっています．

リードのとき，Eの立ち下がりでマシンサイクルが始まり，$t_{AD}+t_{ACC}$時間のあとでデータが確定し，MPUはデータを読み込みます．ですから，t_{ACC}（アクセスタイム）は重要です．t_{ACC}が大きいと，データが準備されなくてもMPUは読み込みを始め，間違ったデータでもリードしてしまいます．アクセスタイムはじゅうぶんに気を付けましょう（**図5-23**）．

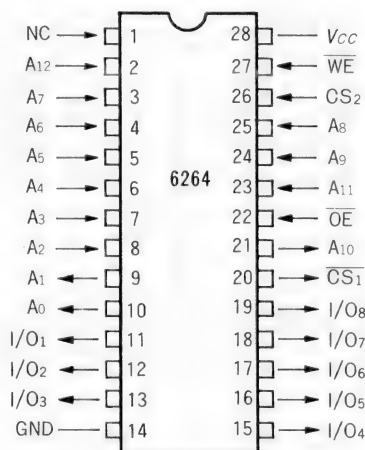



図5-21 RAM6264のチップ・セレクト端子

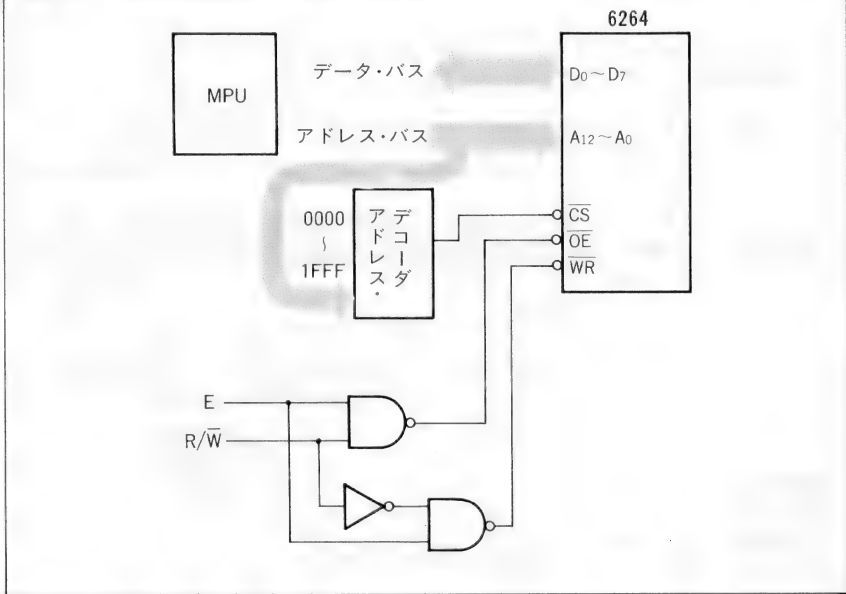



図5-22 $\overline{WR}$, $\overline{OE}$の役目

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	モード
H	×	×	非選択
L	L	H	選択・リード
L	H	L	選択・ライト
L	L	L	選択・ライト

本機採用モード
(×はH, Lどちらでもよいことを示す)

●ROM2732

電源が切れた状態でもデータを保存するためには必ずROM (Read Only Memory)が必要になります．今回，使用した素子は2732と呼ばれる32Kビット（4Kバイト）のEPROM（紫外線でデータを消去できるROM）です．本機ではこのROMを2個使用しています．

1個はシステムの立ち上げ用のブート用，もう1個はシステムを駆動するモニタプログラムの記憶用です．2つのROMはアドレス割り付けの値が異なり

図5-23　6264におけるデータのリードタイミング

リードの時
（Qは省略しています）

クロックE　2.4V　0.5V　2.4V　0.5V

R/$\overline{\mathrm{W}}$　1　0

t_{AD}　t_{ACC}

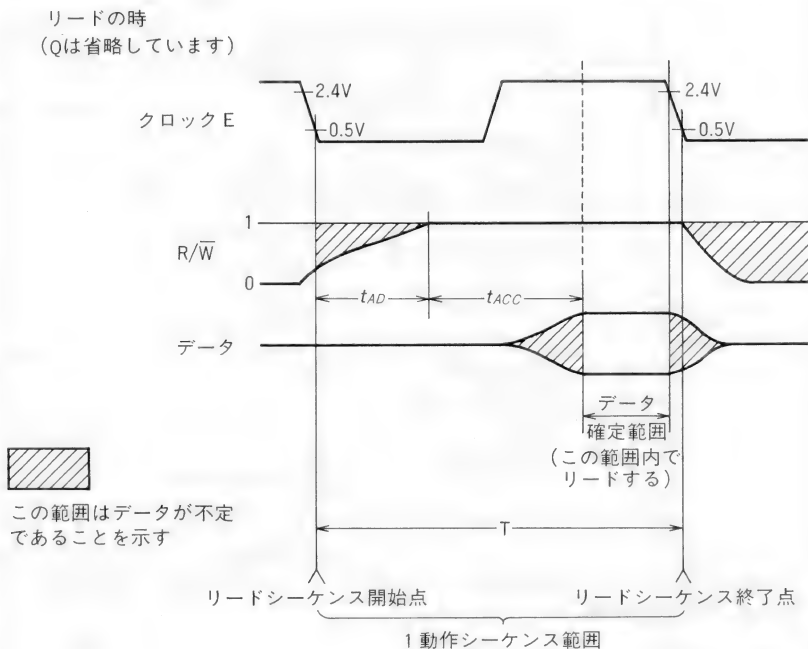
データ

データ確定範囲（この範囲内でリードする）

この範囲はデータが不定であることを示す

T

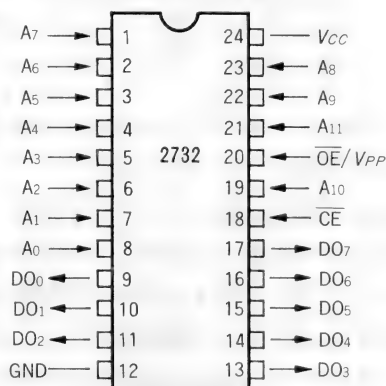
リードシーケンス開始点　リードシーケンス終了点

1動作シーケンス範囲

ますが，アクセス法は同じですので，両方一緒に説明します．

＊$\overline{\mathrm{CE}}$，$\overline{\mathrm{OE}}$の役割

ROM2732のピン接続図は**図5-24**のようになります．チップの選択をする端子は$\overline{\mathrm{CE}}$，$\overline{\mathrm{OE}}$です．

ROMは1個の素子が2つの動作モードを行います．1つはMPUがROMのデータを読むようにする「リード動作モード」，もう1つはROMの中にデータを書き込み記憶させる「ライト動作モード」

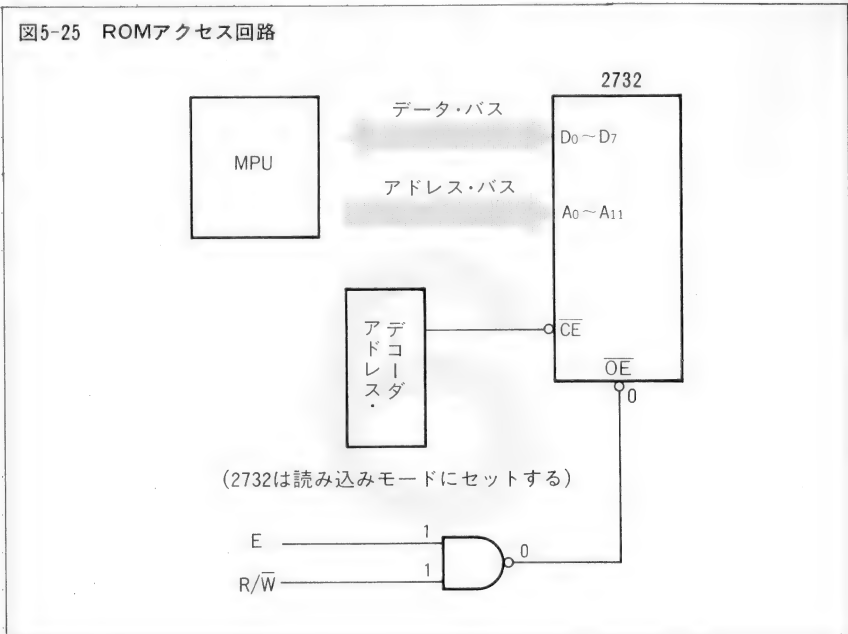
図5-24　ROM2732のピン配置

です．この動作の切り換えをする端子が$\overline{CE}$，$\overline{OE}$です．その動作を**表5-3**に示します．

本書“SBC69の製作”では，ROMへのデータの書き込みを行いません（2巻の「ROMの焼き方」で詳しく述べる）．とりあえずリード動作モードだけでよいので，$\overline{CE}$，$\overline{OE}/Vpp$の両端子を“L”にして動作させます．**図5-25**はROM2732をアクセスする回路です．

ピン名	$\overline{CE}$	$\overline{OE}/Vpp$
読み込みリード	L	L
書き込み	T	＋25 V

表5-3
2732の$\overline{CE}$，$\overline{OE}$の機能

※書き込みモードはSBC69本体上のROMでは使用しない

図5-25　ROMアクセス回路

Chapter Six

I/Oデバイス設計

SBC69はI/OデバイスにPIAを1個，ACIAを1個使用しています．PIAは16進キーボードの接続用，ACIAはホストコンピュータとのデータ通信用のために使います．

I/Oデバイスはわたしたち人間にとってコンピュータと会話をするためにもっとも重要な部分です．この部分が使いやすくなければコンピュータの機能をじゅうぶんに引き出すことはできません．

ここではI/Oデバイスを，回路にどのように組み込んだらよいかということについて解説します．

1 SBC69のI/Oデバイス回路

PIAとACIAの接続方法

SBC69は，I/O デバイスにPIAを1個，ACIAを1個使用しています．PIAは16進キーボードの接続用，ACIAはホストコンピュータとのデータ通信用のために使用します．ここではI/O デバイスを，回路にどのように組み込んだらよいかということについて解説します．

1・1　PIAの接続

PIAはモトローラ系8ビットMPUファミリの代表的LSIです．機能は優れていますが，しかし使い方はそれだけ難しいといえます．PIAにはMC6821を使います（図6-1）．

本機ではPIAの割り込み機能を活用して，キーボードとMPUとの間のデ

図6-1　PIAのピン配置

ータのやりとりをすべて割り込みで処理しています．

ＰＩＡの特徴は，割り込みに対する対応力が高いことです．ＰＩＡのペリフェラル・インターフェース・レジスタに入ってくる信号により容易に割り込みを発生させ，ＭＰＵに割り込み信号が発生したことを知らせます．

＊ＰＩＡの接続回路

ＭＰＵと16進キーボードの接続関係を**図6-2** のように設定しました．図のように，機能を設定してＰＩＡを動作させるためには各ピンをどのように接続すればよいでしょうか．

ＰＩＡはＡポート側とＢポート側の２ポートがありますので，この２つをうまく使い分けるのがＰＩＡを使用するときのコツです．各ピンの接続を**図6-3**に示します．

同図においてRS_0，RS_1という端子に，アドレスのA_0とA_1がそれぞれ接続されていますが，これはＰＩＡ内部のレジスタの選択用に用います．RS_0，RS_1の値と内部レジスタの選択は**表6-1** のようになります．

ここで気づくことは，ペリフェラル・インターフェース・レジスタとデータ方向レジスタが同じ値のRS_0，RS_1により選択されていることです．これはミスプリントではなく，モトローラのＩＣの特徴です．できるだけ少ないアドレス値で，多くのレジスタが選択できるようになっています．詳しくは「７章のＰＩＡのプログラミング」で説明します．

回路においてRS_0，RS_1にアドレスA_0，A_1を接続すると，**表6-2** の関係によりアドレスを指定すれば，希望するレジスタが選択されます．

＊割り込みについて

ＰＩＡには前に述べたように，16進キーボードと発光７セグメントＬＥＤが

RS_1	RS_0	選択されるレジスタ
0	0	ペリフェラル・インターフェース・レジスタA
0	0	データ方向レジスタA
0	1	コントロール・レジスタA
1	0	ペリフェラル・インターフェース・レジスタB
1	0	データ方向レジスタB
1	1	コントロール・レジスタB

表6-1
RS_0, RS_1による内部レジスタの選択

図6-2　MPUと16進キーボードの関係

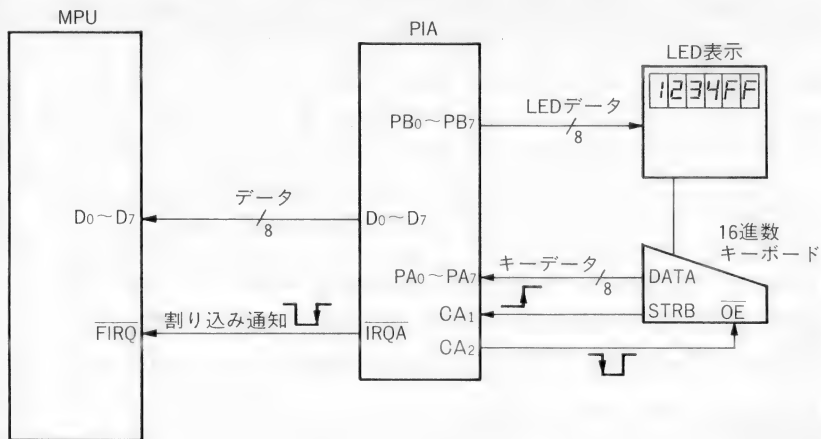



図6-3　PIA各ピンの接続

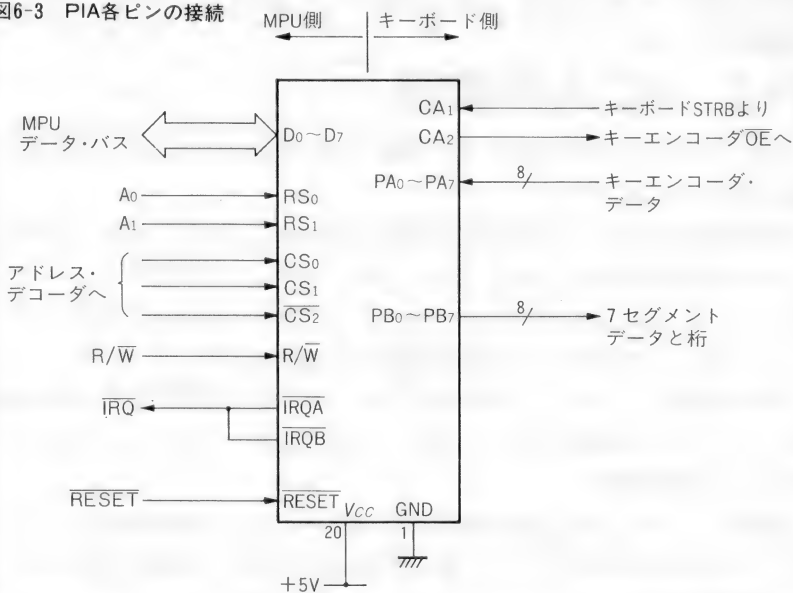



表6-2　アドレスとレジスタの選択

割り付け番地	2進数による番地表現								RS_1	RS_0	選択レジスタ名
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	(A_1)	(A_0)	
$FDFC	1	1	1	1	1	1	0	0	0	0	ペリフェラル・インターフェース・レジスタA
$FDFC	1	1	1	1	1	1	0	0	0	0	データ方向レジスタA
$FDFD	1	1	1	1	1	1	0	1	0	1	コントロール・レジスタA
$FDFE	1	1	1	1	1	1	1	0	1	0	ペリフェラル・インターフェース・レジスタB
$FDFE	1	1	1	1	1	1	1	0	1	0	データ方向レジスタB
$FDFF	1	1	1	1	1	1	1	1	1	1	コントロール・レジスタB

※番地はFDFC～FDFFまで．上位バイトの$FDは共通のため，下位の$FC～$FFに対しての表現

接続されています．そのうちキーボードを押したときのデータはＡポートを通してＭＰＵに伝送され，一方では**ストローブ信号**になってＰＩＡのＣＡ$_1$に伝わります．この信号の立ち上がりエッジで，ＰＩＡの$\overline{IRQA}$が“Ｌ”になって信号が$\overline{FIRQ}$に伝わり，その結果ＭＰＵに割り込みを発生させます．

このためＰＩＡの$\overline{IRQA}$，$\overline{IRQB}$はＭＰＵの$\overline{IRQ}$へ接続しています（$\overline{IRQB}$はＢポートから割り込みを発生させたい場合も想定してＡ，Ｂ側を並列に接続している）．

1･2　ACIAの接続

ＡＣＩＡにはＭＣ6850を使います（**図6-4**）．80系の8251に相当し，直列信号の伝送に使用するＬＳＩです．

本機ＳＢＣ69では，プログラムの開発は別のホストマシンを用いて実行することは５章で説明しましたが，ホストマシンで作成したオブジェクト・プログラム（マシン語に変換されたプログラム）をＳＢＣ69に伝送するときは，このＬＳＩを通して直列データの非同期通信方式で行います．

市販の製品（コンピュータ以外の）でも，たくさん使用されている汎用性の高いＬＳＩです．80系の8251とは違って，使い方はすっきりしています．

＊ＡＣＩＡ接続回路

図6-5に，ＳＢＣ69における6850とホストマシンとの配置関係を示します．ホストマシンのＲＳ232Ｃポートから直列伝送ケーブル（実質は２本のみ）により，ＳＢＣ69の直列信号受信用コネクタ（ＤＢ25使用）を通して，6850のデータ受信用端子ＲＸＤにデータを導きます．

column
コ・ラ・ム

ストローブ信号

システムAからシステムBに信号を転送するときを考えてみましょう。

たとえば、システムAのほうでデータを準備してシステムBに信号を出力したとします。しかし,システムB側のほうではいつデータがくるのかを知る手がかりがないので,データがくるまで待ち続けなければなりません。これではシステムBはほかの仕事が何もできなくなってしまいます。

しかし、もしシステムA側がデータを出力するときにシステムB側のほうに〝データを送ったよ〞という合図を、データとは別の信号として送ってやればどうでしょうか。そうすればシステムB側ではこの合図を待っていればよいだけになるのではありませんか。合図がないときにはシステムBは別の仕事をしていればよいわけです（システムBはこの合図を割り込みとして受け取る)。

このようにデータを出力したという合図の信号をストローブ信号(strobe)といいます。セントロニクスタイプのプリンタポートにおけるストローブ信号とほかの信号とのやりとりを下図に示します。

この図の場合の信号のやりとりは次のようになります。

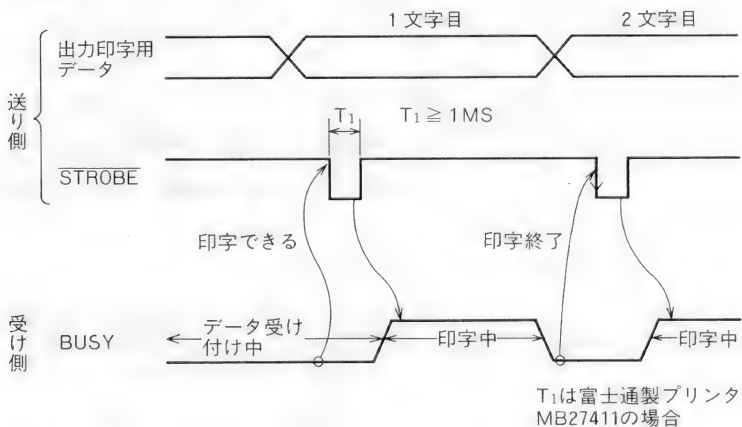
① BUSYが〝L〞になるのを待つ（〝H〞のとき印字中である)
② 8ビットのデータを出力ポートにセットする。
③ STROBEを〝L〞にして1 Mμ後に〝H〞に戻す
④次の文字を印字するときは①に戻る

ここまでの信号は直列信号ですが，6850を用いて並列信号に変換してメモリ内に格納していきます．このような機能を実行できる回路は**図6-6** のようになります．

図6-4　6850のピン配置

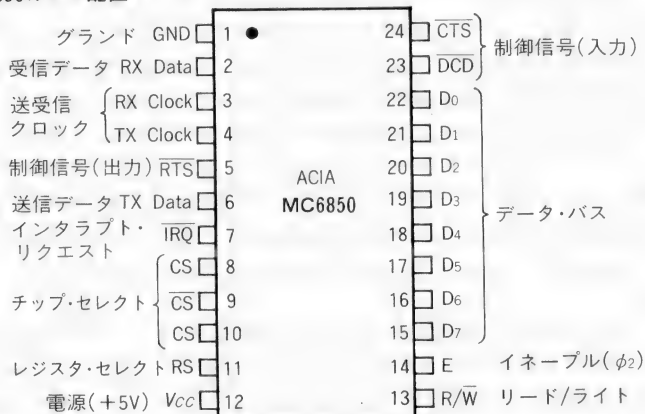
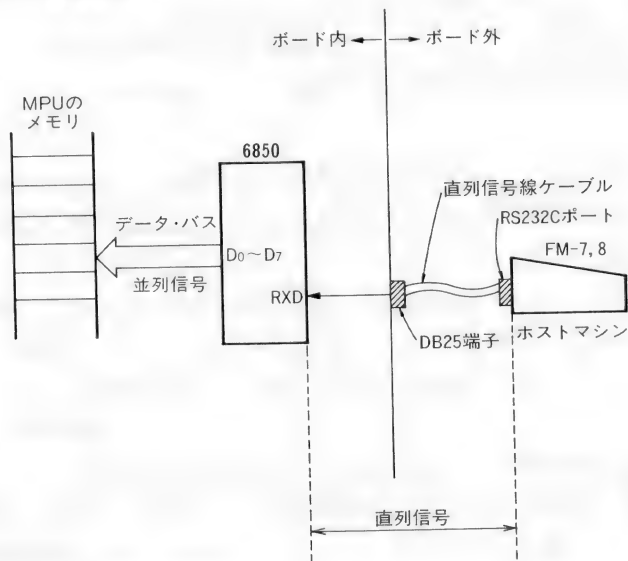
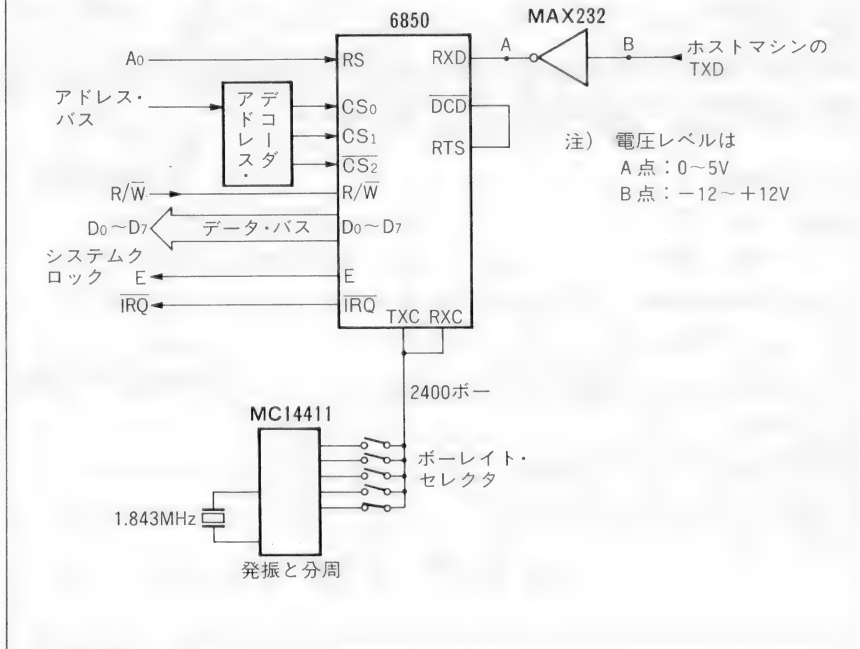



図6-5　6850の配置関係

＊レジスタの選択

ＲＳはRegister Select の略ですが，その名称のとおり6850の内部レジスタを選択するときに使用する端子です．6850は**表6-3** のように，内部レジスタが２つあります．

ステータス・レジスタとコントロール・レジスタ，受信データ・レジスタと送信データ・レジスタは同一アドレスに割り当てられています．それぞれのレジスタを区別する役目をするのがR/$\overline{\text{W}}$端子です．

ステータス・レジスタとコントロール・レジスタは，リードのときがステータスで，ライトのときがコントロールと分けると，１つのアドレスで２つのレ

表6-3
6850の内部レジスタ

R/W ＼ RS	RS＝0	RS＝1
1　（リード）	ステータス・レジスタ	受信データ・レジスタ
0　（ライト）	コントロール・レジスタ	送信データ・レジスタ

ジスタを選別できます．受信データ・レジスタと送信データ・レジスタの場合も同じです．

またＲＳ端子にはアドレス線のA_0が接続されています．そこで割り付けアドレスをＦＤＥ０，ＦＤＥ１とすると，次の関係により，ＦＤＥ０のときはＲＳ＝０，ＦＤＥ１のときはＲＳ＝１になります．

	$A_{15}A_{14}A_{13}A_{12}$	$A_{11}A_{10}A_9A_8$	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$	RSに接続している（A_0）
ＦＤＥ０番地	１１１１	１１０１	１１１０	００００	……→ RS＝0
ＦＤＥ１番地	１１１１	１１０１	１１１０	０００１	……→ RS＝1

以上の関係を**表6-4** にまとめます．このようにＲＳ端子とＲ/$\overline{W}$端子を用いることで，6850の内部レジスタは選択されます．

表6-4
6850内部レジスタの選択

	割り付けアドレス		
	ＦＤＥ０番地	ＦＤＥ１番地	
RS端子 / R/W端子	RS＝0	RS＝1	動作モード
R/W＝1	ステータス・レジスタ	受信データ・レジスタ	リード
R/W＝0	コントロール・レジスタ	送信データ・レジスタ	ライト

入出力機器

データの入力方法と出力表示

コンピュータではMPUが頭脳に相当する働きを示しますが，頭だけでは人間は生きていけません．やはり手や足があってはじめて，人間として行動できます．同じようにコンピュータにおいても，MPUだけでは満足な仕事ができません．人間の手足の代わりである入出力用の機器があってこそ有効な動作ができるわけです．

ここではデータの入力，出力の方法から，それぞれの入出力の機器にまで詳しく解説していきます．

2・1　データの入力方法

コンピュータを作るときにまっ先に考えなければならないことは，何を使って，どのように，どんなタイミングで，データを入力するかということです．SBC69では，２つの入力方法をとりました．

①16進キーボードから入力する

②直列信号用ポートから入力する

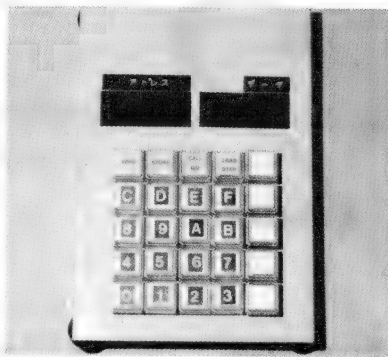
では，この２つの方法から説明していきます．

16進キーボードからの入力

一番オーソドックスなタイプがこの方式でしょう．回路的に簡単な方式では，昔のミニコンのように，スイッチを16個並べて２進値で入力する方法もありますが，現在では非実用的です．やはり一番わかりやすい手法は，この16進キーボードから入力する方法です．

今回使用した16進キーボードは，中古のキーボードを改造して使いました．**写真6-1** は使用するキーボードの外観です．

写真6-1　16進キーボード

なお，このキーボードは，“初歩のディジタル回路シリーズ”の4巻『Z-80実用マイコン製作』で製作したキーボードの仕様とあまり変わらないように工夫しています．そのため，第　　Z-80ワンボード・コンピュータを作られた方はちょっと改造をすることで，本機に合うキーボードに転用することができます．

＊キー構成

キーはデータを入力するデータ用キーと，コンピュータを制御するコントロール用のキーの2種類が必要です．

データ用キーは16進数で入力するのですから，0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F の16個のキーとします．コントロール用キーは多い方がよいのですが，「初歩のディジタル回路シリーズ」4巻で作ったキーボードと同じ仕様にするために，キーの数は4個にします．4巻ではADRS，STORE，GO，INCの4個です．コントロール用キーは限定したわけですが，シリアルポートからの入力用としてLOAD用キーも必要です．しかし，この仕様だとキーが1つ多くなってしまいますから，データやアドレス入力のルーチンを工夫することにして，4巻のキーボードからINCキーを省きました．したがってコントロール用キーはADRS，STORE，GO，LOADの4個になりました．

＊キーボード入力回路のブロック図

キーボードのブロック図は**図6-7** のようになります．キーを1個押すたびにエンコーダ用ICが，どのキーが押されたかを判断して特定のコードに変換します．そして出力端子に出力すると同時にDA端子 (Data Available) が“H”になります．この信号の立ち上がりをストローブ信号としてPIAがキーインの判定に使用します．この部分のタイミングは**図6-8** のようになります．

では，どんなデータがエンコーダ出力から出てくるのでしょうか．エンコーダ出力値の内容を**表6-5** に示します．見てわかるとおり，出力値は2進数の値となって出力されています．なお，表にあるキーポジションとは**図6-9** で示すキーの位置を指します．

本回路では，ストローブ信号の処理はDA端子を直接接続しただけの簡単な

ものです．68系の6821の使いやすいところといえましょう．ただ6821の割り込み検知が立ち上がり，立ち下がりのエッジ・レベル検出なのでこのような簡単な回路ですみますが，80系の8255等になるとレベル検知のためDA信号を，幅

図6-7　キー構成ブロック図

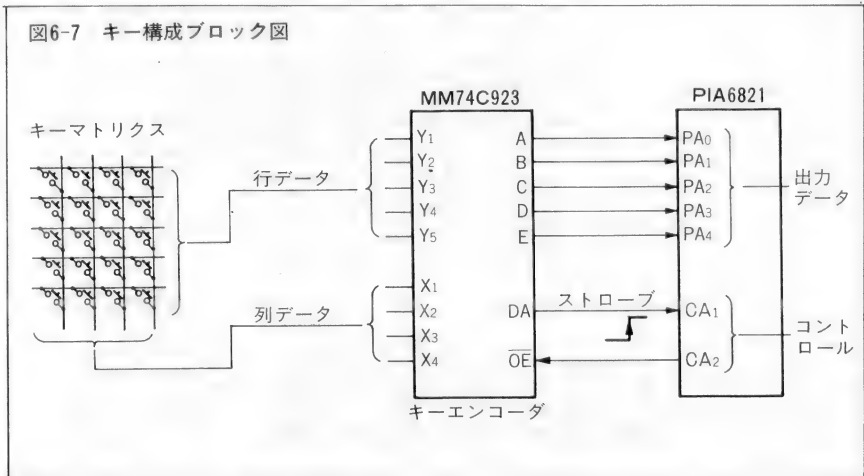



表6-5　エンコーダ出力値

キーポジション	エンコーダ出力値					キー表示値
	E	D	C	B	A	
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	0	0	1	0	2
3	0	0	0	1	1	3
4	0	0	1	0	0	4
5	0	0	1	0	1	5
6	0	0	1	1	0	6
7	0	0	1	1	1	7
8	0	1	0	0	0	8
9	0	1	0	0	1	9
10	0	1	0	1	0	A
11	0	1	0	1	1	B
12	0	1	1	0	0	C
13	0	1	1	0	1	D
14	0	1	1	1	0	E
15	0	1	1	1	1	F
16	1	0	0	0	0	ADRS
17	1	0	0	0	1	STORE
18	1	0	0	1	0	GO
19	1		0	1	1	LOAD

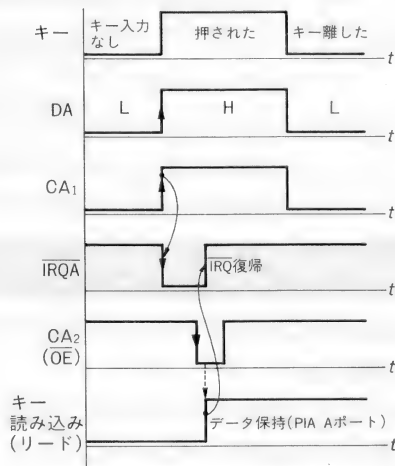
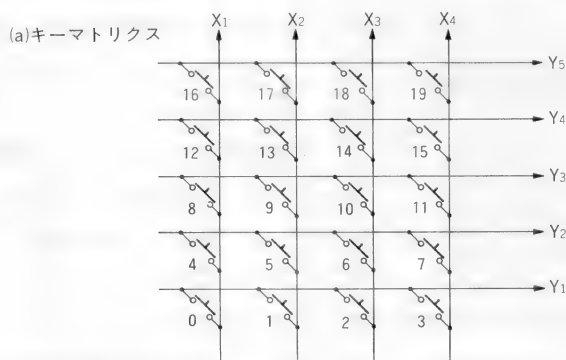
図6-8　キー入力回路のタイミング

※IRQAはデータリードにより復帰する
※CA2="L"にしてエンコーダ出力データをリードする

図6-9 マトリクスとキーポジション

ADRS (10H)	STORE (11H)	GO (12H)	LOAD (13H)
C (0CH)	D (0DH)	E (0EH)	F (0FH)
8 (08H)	9 (09H)	A (0AH)	B (0BH)
4 (04H)	5 (05H)	6 (06H)	7 (07H)
0 (00H)	1 (01H)	2 (02H)	3 (03H)

の狭いパルスに変換する回路が必要です.

RS232Cポートからの入力

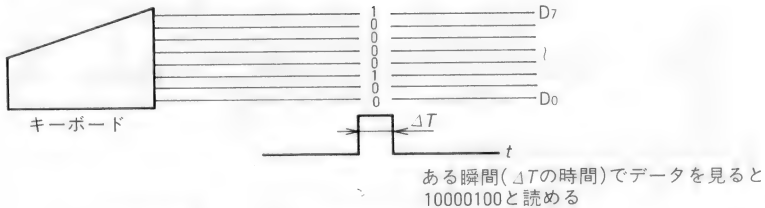
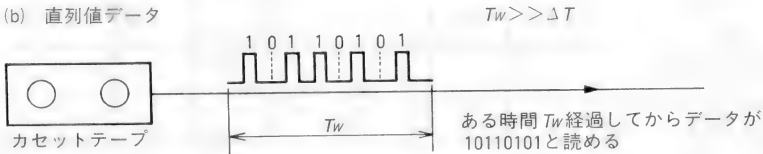
では，もう1つのデータ入力手法であるRS232Cポートからの入力方法について説明します.

コンピュータのデータの型には，次のような2種類があります.

① 並列信号データ

② 直列信号データ

前述のキーボードによるデータは並列信号データであり，カセットからのデータは直列信号データです.（**図6-10**）.

図6-10　データの型

並列信号データは瞬間に読めますが，伝送線路の経費がかかります．一方，直列信号データは伝送線路は最少の費用ですみますが，読み取るのに時間がかかります．２つのデータの形式はお互いに一長一短のデータ形式です．

一般にコンピュータシステムの内部バスの部分は並列信号データを使い，コンピュータのシステム外の部分は直列信号データを使います．

本機はボード内では並列信号データで処理し，ホストコンピュータとの通信回線では直列信号データを用います．

＊直列信号のフォーマット

直列信号はどのような形態をしているのでしょうか．一般的な直列信号フォーマットを**図6-11**に示します．

この直列信号は帯のように連なっていて，どこがデータの始まりで，どこが終わりなのかわかりません．そこでデータの１バイトごとにスタートビット，ストップビットというものを付加して，データの構成をはっきりさせます．これを**非同期式通信方式**といい，よく一般的に使われる方式です．本機もこの方式を採用します（**図6-12**）．

図6-11 直列信号の波形

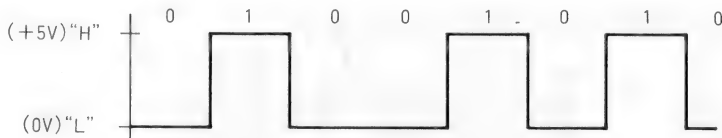
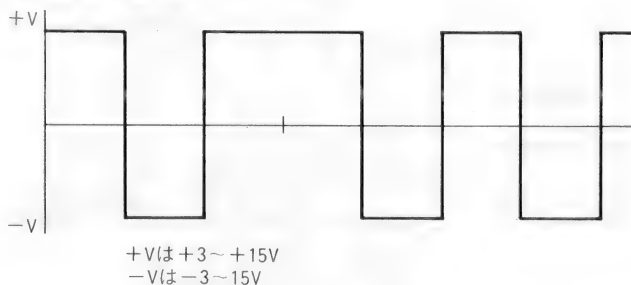
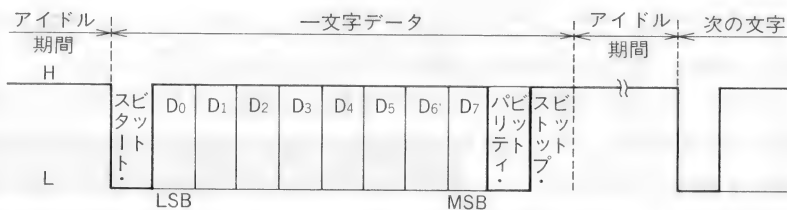
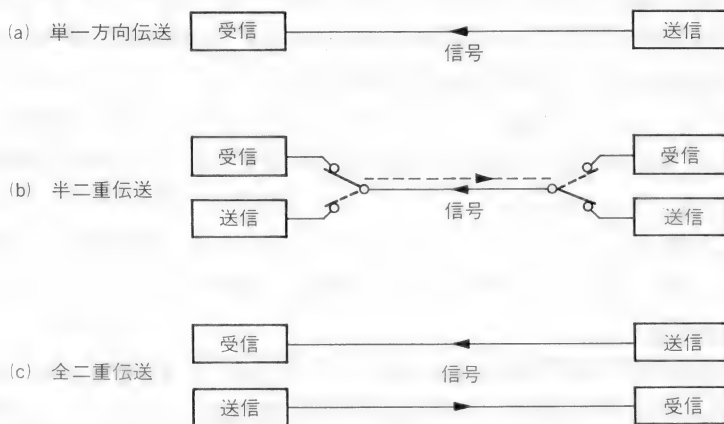



図6-12 非同期式通信のデータ形式

注）① ストップ・ビットは1，1½，2ビットの3種類ある．本機は1ビットとしている。
② データは8ビット，7ビットの2種類があり，本機は8ビットを採用している。
③ パリティ・ビットは付加する，付加しないの2種類があり本機では付加していない。

＊伝送方式はどうなるのか

信号を実際伝送するには各種の方式があります(**図6-13**)．本機では(a) の単一方向伝送方式を用いました．つまり，ホストマシンからＳＢＣ69への一方通行方式です．

ＳＢＣ69からホストマシンへの通信は，ユーザのみなさん方が自由にプログラムを書いて実行してみてください．

Ｓフォーマット・ファイル

ホストマシンからＳＢＣ69へデータを伝送する波形とその方式についてはすでに説明しましたので，ここではホストマシン側からどのようにして伝送し，ＳＢＣ69側でどのように受信するかについて説明します．

ホストマシン側でソース・プログラム（アセンブリ言語で書いたプログラム）を作り，アセンブルして作成したオブジェクト・プログラム（マシン語プログラム）をＳＢＣ69側に伝送するにはさまざまな方式がありますが，あまりに独自の方式で行うと狭い範囲でしか使えない融通性のないマシンになってしまいます．むしろ一般性のある方式を採用したほうがよいでしょう．本機ではモトローラ社が発表しているＳフォーマット方式を採用しています．

Ｓフォーマットは，モトローラ社がM6800を発表したときに，提供した「M

IKBUG」というモニタプログラムで使用した方式です．

この方式は最初は紙テープにデータをパンチしたり，紙テープをロードしたりするときに用いられ，その後も68系のシステムでは直列伝送でデータを送るときの標準方式の1つとして使われてきました．

本機でもこの方式を採用しているので，Sフォーマットでデータ伝送できるコンピュータならどれでも，ホストコンピュータになることができます．

＊レコード・フォーマット

では，**図6-14**にSフォーマットのレコード・フォーマットを見ましょう．本機では，最初の例のレコードタイプS0の様式を使わず，S1とS9の2つのタイプを使用しました．

受ける方では入力されるコードの様子を見ていて，S1というコードがきたら次のデータをバイト数として受け取ります．さらに，次の4バイトをデータ群のアドレスの最初のポインタとして受け取り，後は次からの1バイトずつのデータをバイト数ぶん受け取って，同時に先ほどのアドレス域に格納していきます．そしてS9のフォーマットになったらレコードは終わりであると判定し，

図6-14 Sフォーマット・レコード

1フレーム	ヘッダ・レコード	コード	データ・レコード	コード	EOFレコード	コード
1. スタートオブレコード	53	S	53	S	53	S
2. レコードタイプ	30	0	31	1	39	9
3. 4. バイト数	31 32	12	31 36	16	30 33	03
5. 開始アドレス	31	1	31	1	30	0
6.	30	0	31	1	30	0
7.	30	0	30	0	30	0
8.	30	0	30	0	30	0
9. データ・レコード	34 38	48 “H”	39 38	98	46 43	FC
	34 34	44 “D”	33 32	32		
	35 32	52 “R”				
n. チェックサム	39 45	9E	41 38	A8		

処理を終了します．

実際に受け取ったデータ例は**図6-15**のようになります．

なお，本機ではチェックサム（1フレーム内におけるデータを1バイトずつ単純に加算していき1フレームごとに結果の1の補数をとったもの）の処理をしていません．

本機におけるボートレイトは安全をじゅうぶん見込んで1200ボーとしています（FM-7側は2400ボーが限界）．

＊信号レベルの変換

直列信号を取り扱うときに，1つやっかいな問題があります．それは信号の電圧レベルの統一です．

直列伝送（以後シリアル伝送という）のもっとも基本的な規格であるＲＳ232Ｃでは，ＴＴＬレベルにおける論理1は－3Ｖ～－15Ｖ，論理0は＋3Ｖ～＋15Ｖの間までの振幅が許されています．本機では＋12Ｖ～－12Ｖの両電圧レベルを採用しました．

このためホストコンピュータから伝送される信号からは，＋12Ｖ～－12Ｖの両電圧レベルが送られてきます．しかしこのまま伝送されてきた信号を受信すると，ＡＣＩＡ6850はＴＴＬレベルのＬＳＩなので，信号－12Ｖ～＋12Ｖの電圧レベルの信号を受信すると，6850が破壊されてしまいます．ゆえに，**図6-16**

図6-15　Sフォーマット・データの例

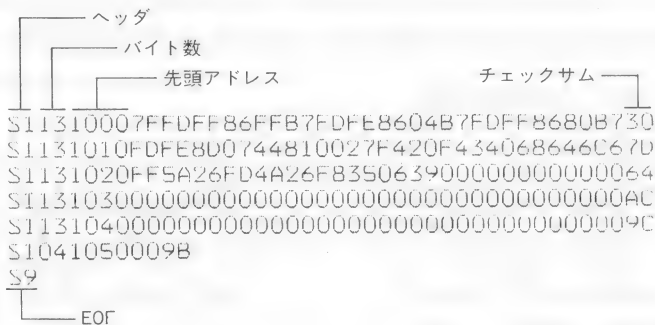
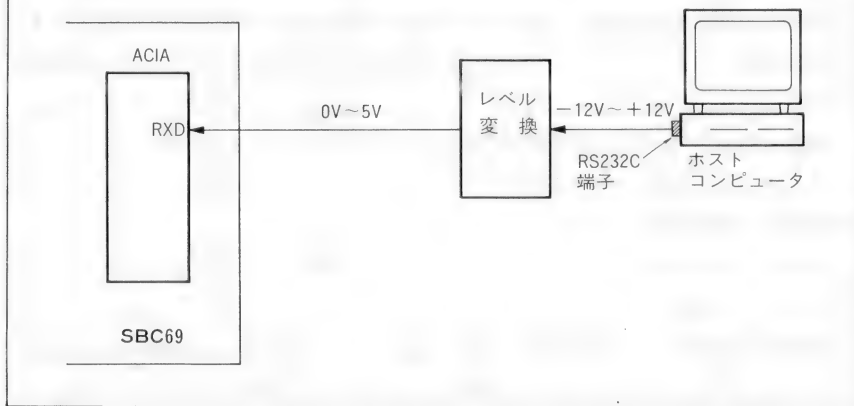



図6-16 シリアル・データの電圧レベルの変換

のような電圧レベル変換機能を，ＳＢＣ69とホストマシンの間に設置する必要があります．

＊電圧レベル変換用ＩＣ

このようなＴＴＬ電圧レベルと RS232C 電圧レベルとの間に使用するのが，電圧レベル変換用ＩＣです．

代表的なものにＳＮ75188 ，ＳＮ75189 Ａのペアがあります．ほかにはＭＣ1488，ＭＣ1489なども同じ内容です．

75188 はＴＴＬ電圧レベルを RS232C レベルに変換し，75189 は±12Ｖの RS232C レベルを０～５ＶのＴＴＬレベルに変換します．２つのＩＣのピン接続は**図6-17**のようになります．**図6-18**は，本回路にこのＩＣを使用したときの

図6-17 75188，75189の端子接続

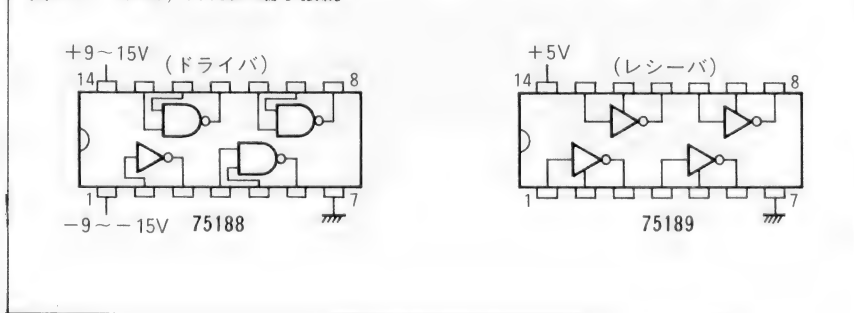
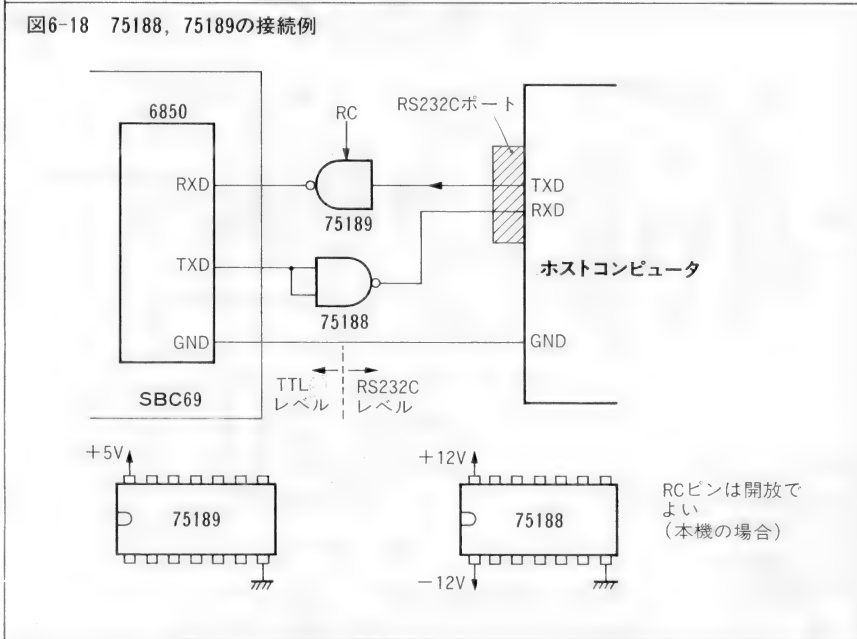



図6-18　75188，75189の接続例

接続例です．

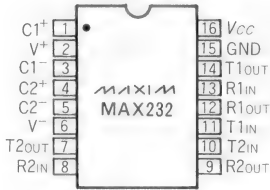
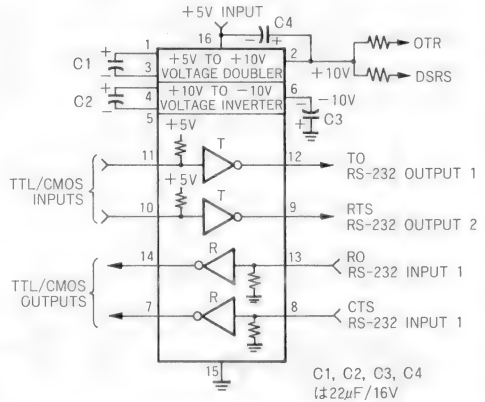
この２つのＩＣは価格が安く，どこでも購入できる一般的な製品ですのでお勧めします．しかしＳＢＣ69の回路をよく見てみると，使用しているものはＭＡＸ232 という見慣れないＩＣです．

なぜ，ＳＢＣ69では前述の75188 ，75189の両ＩＣを使用しなかったのでしょうか．この理由は電源を簡素化するために，＋５Ｖの１つの電源でＳＢＣ69を動作させたかったからです．

75188 ，75189 は安価で確かに便利なのですが，±12Ｖの電源が必要になってきて，取り扱いや処理が非常に面倒になってきます．その点，本機で使用したＭＡＸ232 を使えば簡単にコンパクトな装置を作ることができます．ただし発表してからまだ期間がたっていないので，現在では若干値段が高いのが欠点といえましょう．

図6-19にＭＡＸ232 の各種データを示します．このＩＣは現在の軽薄短小化の要請に合っているため，今後もかなり多く使われるものと思います．

図6-19　MAX232の特性

(1)ピンの配置

(2)ピンの接続

(3)種類

PART	TEMP.RANGE	PACKAGE
MAX232CPE	0℃to+70℃	16 Pin Plastic DIP
MAX232CWE	0℃to+70℃	16 Pin Small Outline
MAX232C/D	0℃to+70℃	Dice
MAX232EPE	−40℃to+85℃	16 Pin Plastic DIP
MAX232EWE	−40℃to+85℃	16 Pin Small Outline
MAX232EJE	−40℃to+85℃	16 Pin CERDIP
MAX232MUE	−55℃to+125℃	16 Pin CERDIP

(4)絶対最大定格

V_{CC} ……………… 6V
V^+ ……………… 12V
V^- ……………… 12V
◎入力電圧
$T1_{IN}$、$T2_{IN}$
……………… −0.3−(V_{CC}−0.3V)
$R1_{IN}$、$G2_{IN}$
……………… ±30V
◎出力電圧
$T1_{OUT}$、$T2_{OUT}$
……………… (+0.3V)～(V−0.3V)
$R1_{OUT}$、$R2_{OUT}$
……………… −0.3～(V_{CC}+0.3V)
◎短絡時間
V^+ ……………… 30秒
V^- ……………… 30秒
$T1_{OUT}$、$T2_{OUT}$ ……………… 連続
◎消費電力
CERDIP ……………… 500mW
70℃以上9.5mW/℃の減定格
Plastic DIP ……………… 375mW
70℃以上7mW/℃の減定格
Small Outline (SO) ……………… 375mW
70℃以上7mW/℃の減定格

2・2　データの出力表示

コンピュータがデータを受け取って（プログラムもデータである）処理が終了したら，その結果をどのようにして人間に知らせたらよいでしょうか．

一番簡単な方法は，ＬＥＤを８個並べて（データの場合８ビットでよいが，アドレスも表示するとさらに16個必要），２進数の値で表示する方法です（**図6-20**(a)）．もう１つの方法は，０からＦまでの数値を発光表示できる７セグメント発光ＬＥＤを使って，16進数で表示する方法です（**図6-20**(b)）．

16進数表示の方が私たちにとって理解しやすいことは説明するまでもないでしょう．本機では16進数表示の方法を採用しています．

＊７セグメントＬＥＤによるデータ表示

７セグメントＬＥＤによる表示法は電卓のディスプレイなどでおなじみなように，現在もっともポピュラーな表示手段の１つです．一般的なＬＥＤの発光表示とセグメント点灯の位置関係は**表6-6** のようになります．

つまり７セグメントによるデータ表示は，７個（少数点も入れると８個）ある発光ダイオードをどんな位置関係で点灯するかということなのです．１個ずつのＬＥＤ（これをセグメントという）は**図6-21**のような接続で点灯します．このような単体のＬＥＤを適当に組み合わせることにより，種々のパターンが表示できます．ＰＩＡとの接続は**図6-22**のようになります．

＊７セグメントＬＥＤの種類

注意することは，７セグメントＬＥＤには２つの種類があり，この選択を誤ると発光しないことです．これはカソード側を共通にまとめるか，アノード側を共通にまとめるかによって，点灯するための電圧の極性が逆になるためです．

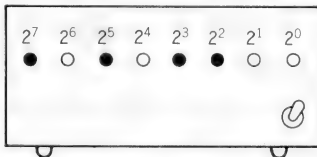
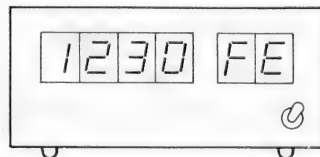
図6-20　データの出力表示方法

(a)　LEDによる２進表示　　(b)　７セグメントLEDによる16進表示

表6-6
表示とセグメントによる点灯の位置関係

0	1	2	3
4	5	6	7
8	9	A	b(B)
C	d(D)	E	F

図6-21 LEDの構造と発光回路

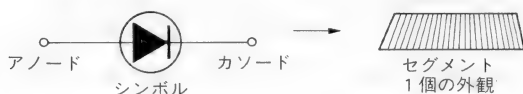
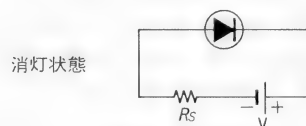
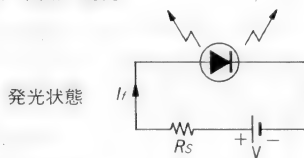



図6-22　PIAとの接続関係

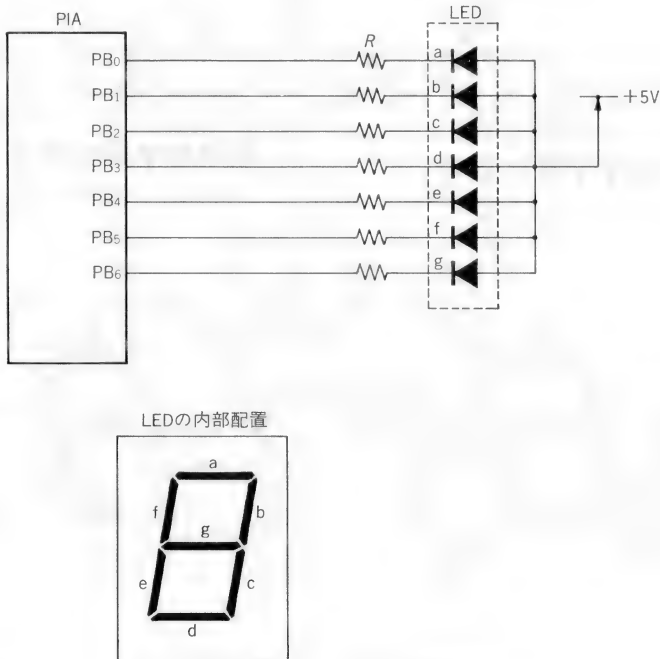



図6-23中の(a) のタイプをアノードコモン，(b) のタイプをカソードコモンと呼びます．本機ではアノードコモンの素子を用いました．

＊使用する７セグメントＬＥＤについて

最近のＬＥＤは高輝度になり，電流消費も少なくなってきています．本機で使用する素子もそのタイプで，非常に明るくくっきりと数字が明示されます．ＳＢＣ69では，シャープ製のＧＬ７Ｄ201 とＧＬ７Ｅ201 の製品を使用することにしました．

前者は赤色発光の高輝度タイプ，後者は黄緑色発光の高輝度タイプです．各種素子特性の一覧表は巻末に掲載してあります．

ＳＢＣ69では７セグメントＬＥＤを**図6-24**のように配置しました．データ表示とアドレス表示は色を別にしておくと識別に困りません．７セグメントＬＥ

図6-23　7セグメントLEDの種類

図6-24　7セグメントLEDの配置

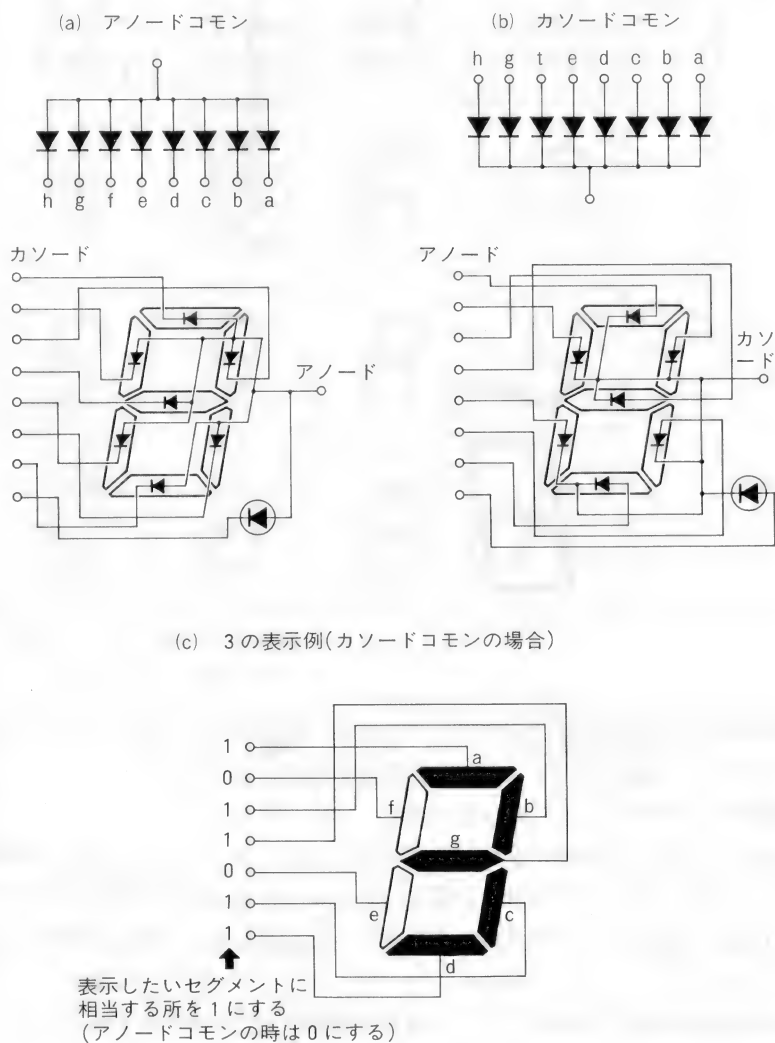
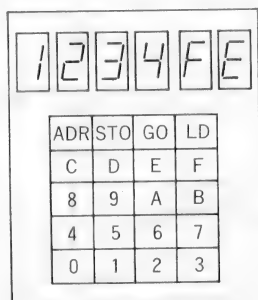
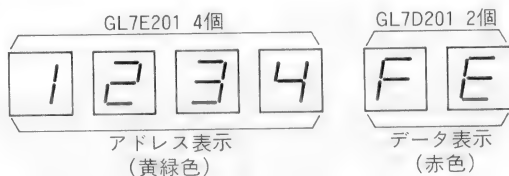
GL7E201 4個　　GL7D201 2個

1 2 3 4　F E

アドレス表示
（黄緑色）

データ表示
（赤色）

1 2 3 4 F E

ADR	STO	GO	LD
C	D	E	F
8	9	A	B
4	5	6	7
0	1	2	3

16進キーボードの外観

Dは**写真6-2** のようになります.

＊表示方式をどうするか

７セグメント素子自体に２つの種類があったように，ＬＥＤのデータ表示方式にも次の２種類があります.

①スタティック表示方式

②ダイナミック表示方式

①，②の表示方式の概略は**図6-25**のようになります.

①のスタティック表示方式は，１つ１つの発光素子に専用の表示パターン用データを送り込む形式です．この方式は，データ用の線が７本×６＝42本も必要になるの

写真6-2　7セグメントLED

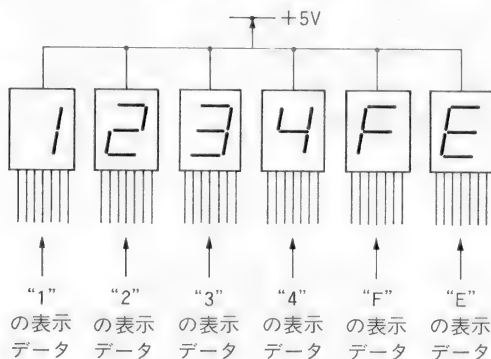
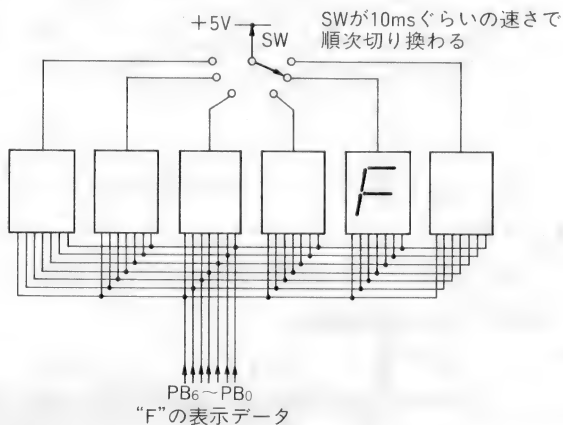
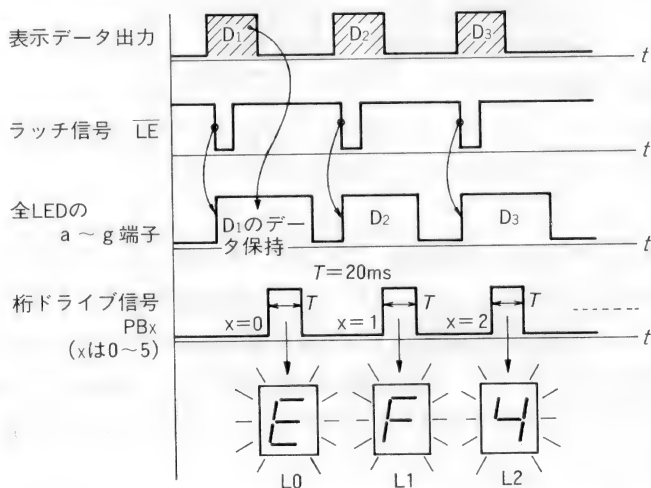
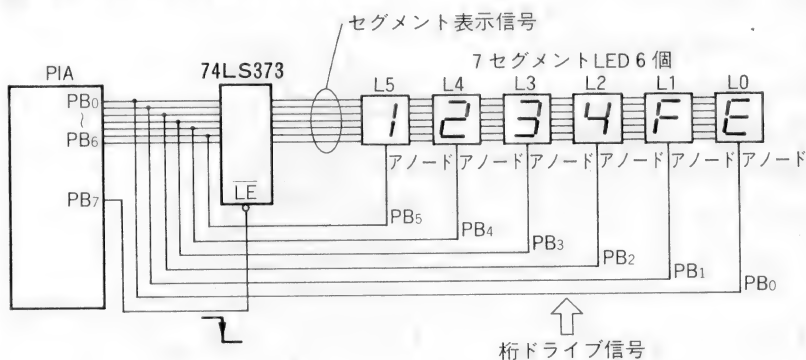



図6-25　2つの表示方式

で大変です．また素子１個ごとに別々のデータを送り込むために表示データ転送用回路が６個必要となり，このような簡単な表示器には不向きです．しかし，この方式は，たとえコンピュータが止まっていても，表示数値はスタティック（静的）に安定して光っています．この状態を指してスタティックという名称になりました．

ＳＢＣ69ではダイナミック表示方式にしています．７セグメントＬＥＤ表示回路と表示タイミングは**図6-26**のようになります．

図6-26　7セグメントLEDドライブ回路

②のダイナミック表示方式は，表示数値は1個の表示器に安定して静止して表示されているのではなく，20m秒くらいの短い時間のみ表示され，すぐに次の点灯表示に移っていく方式です．ダイナミック（躍動的）に表示が変化している様子から，ダイナミック表示方式といわれています．

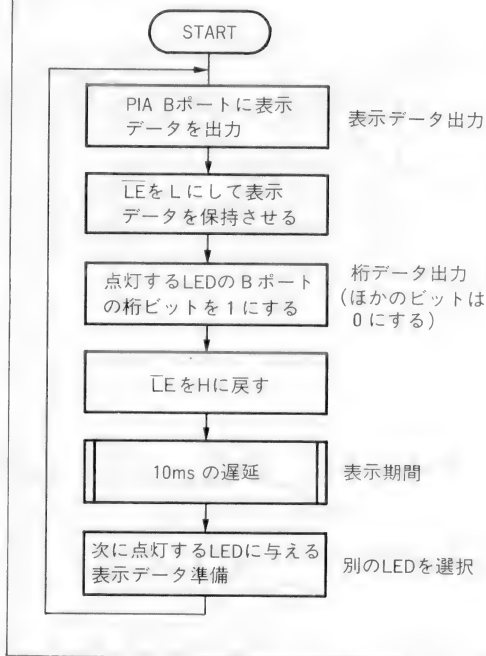
人間の目は残像作用を持っているため，表示はどんどん移動しながら表示されているのですが，私たちにはあたかも静止して数値が表示されているように見えるわけです．ただ，輝度はスタティック方式と比較すると相当に低下しま

す．しかし，経済性と回路の簡略化が簡単ということから，ほとんどの機器はこの方式を採用しています．

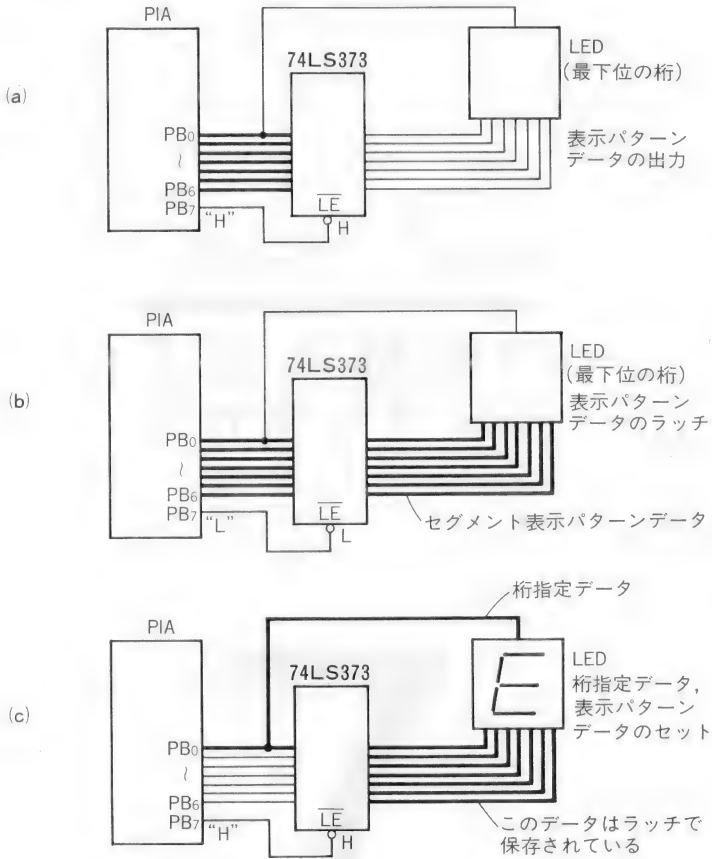
＊桁と表示データの分類

図6-26の回路を見ると，ＰＩＡ6821のＢポートから，桁用データと表示用データの両方が出力されているのがわかりますね．この２つのデータをどうやって分離したらよいのか考えてみましょう．

74ＬＳ373 はラッチと呼ばれる機能ＩＣで，入力したデータを内部に保持するＩＣです．データ保持のタイミングはＬＥ端子が“Ｈ”から“Ｌ”になった瞬間です．

図6-27　フローチャート

この手順をフローチャート（流れ図）に示すと，**図6-27**のようになります．具体的に信号の流れを時間的に追って見ていくと，**図6-28**のようになります．(b) の期間に，もしＰB_0が１の状態ですと，表示データをラッチするときに一瞬点灯する可能性がありますが，非常に短い時間ですから，人間の目には見えません．そして，(c) の段階で正規の桁データが与えられ正しく点灯し続けることになります．

なお，桁データはＬＥＤがアノードコモンを使用しているので，“Ｈ”レベルを与えると点灯します（Ｈ論理）．

図6-28　点灯までのデータの流れ

Chapter Seven

入出力機器のプログラミング

モニタプログラムの作成(第2巻)において使用される入出力用機器にはPIAとACIAというLSIがあります．本章ではこの2つのLSIを中心に解説していきます．

入出力LSIは機械を制御するときに直接に制御対象とデータのやりとりをする大事な主役です．MPUが頭脳とすれば，入出力用LSIは人間の手足といえるでしょう．しかし，構造的にはMPUよりもわかりにくくなっています．MPUは論理的に追いかけていくことで理解できるのですが，入出力用LSIは感覚的な面が多く，想像力や推理力が必要とされます．

第2巻で登場してくるモニタプログラムのソースリストの理解においては両LSIに対する知識がじゅうぶんに必要となります．とくにPIAは68系ファミリの中心的LSIです．PIAを完全に理解できればほかのファミリLSIの理解は容易です．たくさんの表が出てきて大変ですが，もう少しの辛抱です．実際にマシンが動いてくれればそれまでの疲れは吹き飛ぶことでしょう．

1 PIAのプログラミング

コントロール・レジスタ データ方向レジスタ

ＰＩＡはMPUと入出力機器との間をインターフェースします．本機ではＰＩＡに68系ファミリのＬＳＩであるMC6821Aを使用します．

機能的には柔軟性がある入出力専用のＬＳＩです．原則として並列データを取り扱います．ＰＩＡの機能はプログラムで設定されます．したがってＰＩＡをドライブするプログラムを理解しておかないと動作させることはできません．

ＰＩＡの全機能はシステムをイニシャライズするときに決定されます．イニシャライズとは，ＭＰＵからＰＩＡの内部レジスタであるコントロール・レジスタに初期設定用データを書き込むことをいいます．

1・2　PIAの特徴

まず，ＰＩＡの特徴から説明しましょう．ＰＩＡには次のような特徴があります．

・入出力機器にインターフェースできる２組の８ビット双方向性のペリフェラル・インターフェース・レジスタを持つ
・プログラム可能な２本のコントロール・レジスタを持つ
・プログラム可能な２本のデータ方向レジスタを持つ
・単独で使用できる割り込みラインが４本ある
・入出力機器制御のためのハンドシェイク機能がある
・Ａ，ＢポートともにＴＴＬは２個，ＬＳＴＴＬは８個をドライブできる
・割り込みはＰＩＡ側でマスクできる
・入出力の方向はデータ・バスの各ビットごとに設定できる
・割り込みは入力パルスのエッジにより発生する（80系の8255はＬレベルによるレベル検出タイプ）

・4本の独立したフラグ制御入力線が用意されている
・入力ポートに設定したときにプルアップ抵抗が内蔵されている
・出力データはラッチされ，保持される

1・2　PIAの端子

ＰＩＡのピン配置は6章の図6-1でも示しましたが，再度図7-1に示します．またＰＩＡの内部構成は図7-2のようになります．

①双方向データ・バス（D_0～D_7）

ＭＰＵとＰＩＡ間でデータのやりとりをするときに使用します．双方向性で直接ＭＰＵに接続することができます．

②ペリフェラル・データ・バス　Ａポート（PA_0～PA_7）

ＰＩＡと入出力制御機器とがデータのやりとりをするときに使用します．各ビットごとに自由に入力用，出力用に設定ができます．

ＰＡｎ（ｎは０～７）の各端子は常に入力ポートに接続されていますので，出力端子に設定したときも入力ポートからＰＡｎ端子のデータを読み取ることができるので非常に便利です．しかし，ＰＡｎ端子に大きな負荷を接続したときは論理レベルが正しくなり，出力値と異なった値を読み込みますので注意しなければなりません．なお，ペリフェラル・インターフェース・レジスタと

図7-1　PIAのピン配置

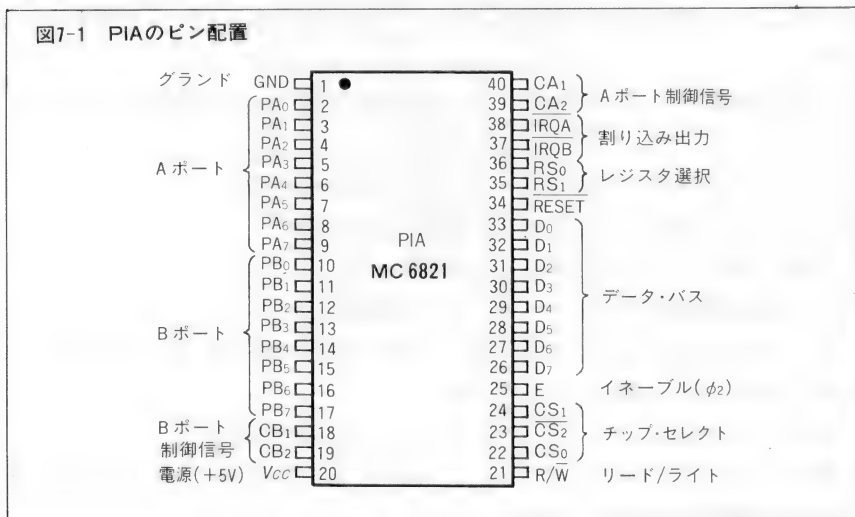
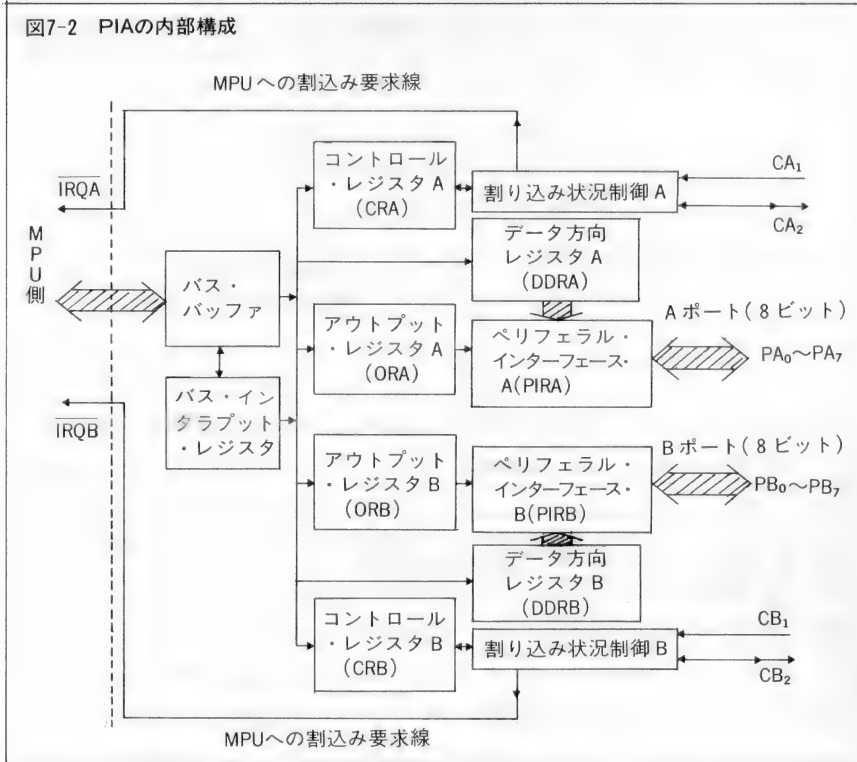



図7-2　PIAの内部構成

はアウトプット・レジスタとペリフェラル・インターフェース・バスを総称した名前です．Aポート側の特徴は出力端子に（入力のときも）約６ＫΩのプルアップ抵抗 が内蔵されている点です（**図7-3**）．

③ペリフェラル・データ・バス　Bポート（PB_0～PB_7）

Aポート側と同様に，ＰＩＡと入出力制御機器間でデータのやりとりをするときに使用します．Bポート側の構造は**図7-4**のようになります．

Aポート側と異なる点はプルアップ抵抗がなく，出力端子がスリーステートになっていることです．そのためBポートを入力に設定したときはハイインピーダンス状態になります．

図7-4の等価回路からわかるように，出力レジスタからのデータはすぐに入力用バッファのほうへ戻っています．したがって出力端子ＰＢｎの状態に関係なく(PBnがショートしたとしても)，出力されたデータの再読み込みができます．

図7-3　ペリフェラル・データ・バスA側の等価回路

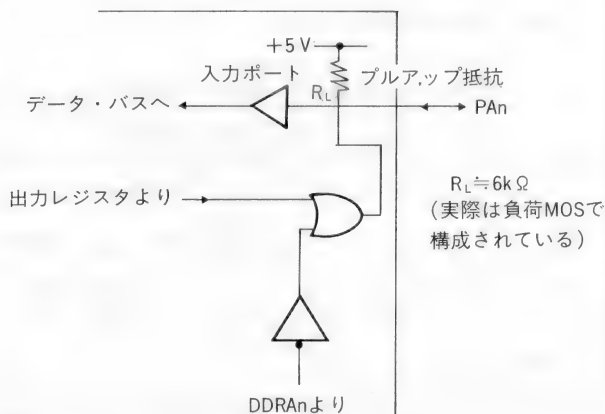
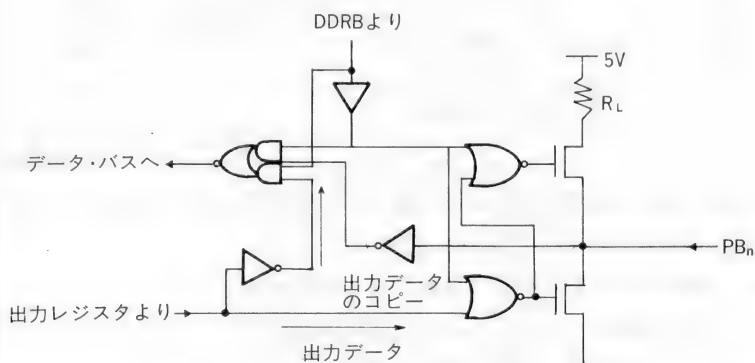



図7-4　ペリフェラル・データ・バスB側の等価回路

④チップ・セレクト（CS_0，CS_1，$\overline{CS_2}$）

入力端子です．MPUのアドレス・バスと組み合わせてPIAを選択するのに使用します．CS_0，CS_1には“1”を，CS_2には“0”を与えることで論理レベルが満足します．PIAを選択するときは必ず正しい論理レベル値を与

RS_1	RS_0	選択されるレジスタ
0	0	ペリフェラル・インターフェース・レジスタA (PIRA)
0	0	データ方向レジスタA (BDRA)
0	1	コントロール・レジスタA (CRA)
1	0	ペリフェラル・インターフェース・レジスタB (PIRB)
1	0	データ方向レジスタB (DDRB)
1	1	コントロール・レジスタB (CRB)

表7-1
RS_0, RS_1による内部レジスタの選択

える必要があります．

⑤レジスタ・セレクト（RS_0，RS_1）

ＰＩＡの内部の６個のレジスタを選択するときに使用します．選択の様子は５章**表5-2**で示しましたが，ここではもう一度**表7-1**に示します．一般にRS_0にアドレス線のA_0を，RS_1にA_1を接続します．

⑥インタラプト・リクエスト（$\overline{\mathrm{IRQA}}$，$\overline{\mathrm{IRQB}}$）

出力端子です．ＭＰＵの割り込み要求端子$\overline{\mathrm{IRQ}}$や$\overline{\mathrm{FIRQ}}$に接続され，割り込み動作を行います．割り込み要求がＰＩＡの回路で生じると“Ｌ”レベルになり，ＭＰＵに割り込みを要求します．

この２つの割り込み端子はオープンドレインで引き出され，ワイアードＯＲですから，多数のＰＩＡと並列に接続できます．

割り込み要求がＰＩＡに接続されている機器から発生しても，$\overline{\mathrm{IRQA}}$，$\overline{\mathrm{IRQB}}$が必ずしも“Ｌ”にならなくてもよい場合にはプログラムでマスクすることができます．この場合には$\overline{\mathrm{IRQA}}$，$\overline{\mathrm{IRQB}}$が“Ｌ”になることはできません．

⑦ペリフェラル・コントロール・バス（CA_1，CA_2，CB_1，CB_2）

４つの端子はそれぞれ異なった機能を持っています．これらの端子の使い方は複雑であり，ＰＩＡを理解しにくくしている一因といえるでしょう．

・CA_1，CB_1

この２つの端子は入力機器からの割り込みに対応して，コントロール・レジスタのインタラプト・フラグをセットするための割り込み入力端子です．割り込みのアクティブ・エッジをプログラムにより，立ち上がり，立ち下がりのどちらかに設定できます．

・CA_2

この端子は割り込み入力として，または1ビットの出力端子としての両方にプログラムで設定できます．

・CB_2

CA_2と同様に割り込み入力として，または1ビットの出力端子としてプログラムで設定できます．ただCB_2はドライブ能力が強化されていて，トランジスタのベースを直接ドライブする能力（1.5Vで2.5mA）を持っています．なお，この4つの端子の詳しい使い方はあとで説明します．

⑧イネーブル端子（E）

PIAの内部タイミング用信号を入力する端子です．PIAの動作はこのパルスに同期して実行されます．MPU6809のシステムクロックEに接続します．

⑨リード・ライト端子（$R/\overline{W}$）

MPUとPIA間におけるデータ転送方向の制御信号を入力します．この端子が“L”レベルのときにはMPUからPIAの方向に，“H”レベルのときにはPIAからMPUの方向にそれぞれデータが転送されます．

⑩リセット端子（$\overline{RESET}$）

PIAのリセットを行います．この端子は最低1μsecの期間“L”にすることでリセットが実行されます．そのとき，すべての内部レジスタは0になり，次のような状態となります．

1) ペリフェラル・インターフェース・レジスタはすべて入力として設定される
2) CA_1，CA_2，CB_1，CB_2は立ち下がりエッジでアクティブに設定される
3) 全部の割り込みはマスクされるゆえに割り込みフラグのビットがセットされても$\overline{IRQA}$，$\overline{IRQB}$は“L”にならない．

1・3　コントロール・レジスタ

PIAのプログラミングではコントロール・レジスタの働きを完全に理解しておかないと先に進めません．このレジスタを理解できるかどうかがPIAのプログラミング克服の決め手です．

コントロール・レジスタCRA，CRBはPIAのステータスを表す部分と，コントロール用データの部分に分かれます（**表7-2**）．CRAとCRBは基本的には同じですが，ハンドシェイクモードのときには両者は違った働きをします．

表7-2　コントロール・レジスタのビット割り付けと機能

	ビット位置	7	6	5	4	3	2	1	0
CRA	区分	ステータス	ステータス	コントロール	コントロール	コントロール	コントロール	コントロール	コントロール
	機能名	割込みフラグ IRQA1	割込みフラグ IRQA2	CA_2制御			DDRA選択	CA_1制御	

	ビット位置	7	6	5	4	3	2	1	0
CRB	区分	ステータス	ステータス	コントロール	コントロール	コントロール	コントロール	コントロール	コントロール
	機能名	割込みフラグ IRQB1	割込みフラグ IRQB2	CB_2制御			DDRB選択	CB_1制御	

※ステータス・ビットはリード・オンリーでライトはできない．

つまりこのモードのときにはＡポートは入力モードに，Ｂポートは出力モードとして働きます．

コントロール・レジスタの各ビットごとの機能は**表7-3**　**表7-4**のようになります．

●CA_1，CB_1の機能設定

では，ビット０とビット１が行うCA_1，CB_1の制御について説明します．CA_1，CB_1は割り込み要求入力パルスのエッジが立ち上がりか,立ち下がりなのかを決めるビットです．

割り込み入力パルスのエッジの方向を決定するものは，ＡポートＡ用ではＣＲＡ０（コントロール･レジスタのビット０）とＣＲＡ１（同じくコントロール･レジスタのビット１），Ｂポート用ではＣＲＢ０とＣＲＢ１です．また割り込みフラグのセット機能もあります（**表7-5**）．

ＣＲＡ７(ＣＲＢ７)の割り込みフラグはＣＲＡ０,ＣＲＡ１によりセットできますが,ＣＲＡ７はステータス･ビットなので書き込みはできません.このためＣＲＡ７は一度セットされてしまうと,コントロール･レジスタに対して直接０の書き込み等による方法ではクリアすることはできないことになります．このフラグはMPUがＰＩＡのペリフェラル・インターフェース・レジスタを読み込むときにクリアされます．そこで，このフラグをクリアするためだけに，ペリフェラル・インターフェース・レジスタをリードすることがあります．これは「空読み」といって，ＰＩＡ独特のテクニックです．ただし，この場合Ｃ

表7-3　コントロール・レジスタA(CRA)

CRA		ビット		0	1
0	CA_1制御	CRA0 (CA_1割込み出力マスクビット)		ペリフェラル・コントロールライン CA_1 による割込み信号によりCRA7($\overline{IRQA_1}$)はセットされるが，$\overline{IRQA}$はマスクされ出力されない("High"のまま)。	ペリフェラル・コントロールライン CA_1 による割込み信号によりCRA7($\overline{IRQA_1}$)がセットされると$\overline{IRQA}$出力が"Low"になる.
1		CRA_1(CA_1アクティブエッジビット)		CRA7($IRQA_1$)をCA_1入力の立ち下がりでセット.	CRA7($IRQA_1$)をCA_1入力の立ち上がりでセット.
2	DDRA	CRA2		データディレクション・レジスタAを指定.	ペリフェラル・インターフェースレジスタAを指定.
3	CA_2制御	CRA3 (CA_2割込み出力マスクビット)	CA_2入力指定のとき	ペリフェラル・コントロールライン CA_2 による割込み信号によりCRA6($\overline{IRQA_2}$)はセットされるが，$\overline{IRQA}$はマスクされ出力されない("High"のまま).	ペリフェラル・コントロールライン CA_2 による割込み信号によりCRA6($\overline{IRQA_2}$)がセットされると$\overline{IRQA}$出力が"Low"になる.
4		CRA4(CA_2アクティブエッジビット)	CA_2入力指定のとき	CRA6($IRQA_2$)をCA_2入力の立ち下がりでセット.	CRA6($IRQA_2$)をCA_2入力の立ち上がりでセット.
3		CRA3 ハンドシェイクモード指定ビット	CA_2出力指定のとき	CRA4＝0のとき ペリフェラル・コントロールライン CA_2 出力はMPUがペリフェラル・インターフェース・レジスタAを読出したときEパルスの立ち下がりで"Low"となる. ペリフェラル・コントロールライン CA_1 信号のアクティブエッジで"High"となる.	CRA4＝0のとき ペリフェラル・コントロールラインCA_2出力はMPUがペリフェラル・インターフェース・レジスタAを読出したときのEパルスの立ち上がりで"Low"となる. 読出し動作後，次のEパルスの立ち下がりで"High"となる.
4		CRA4	CA_2出力指定のとき	CRA3をハンドシェイクモード指定ビットとして有効にする.	CRA3の値をそのままCA_2に出力する.
5		CRA5(CA_2入出力プログラムビット)		ペリフェラル・コントロールライン CA_2を入力に指定.	ペリフェラル・コントロールライン CA_2 を出力に指定.
6	$IRQA_2$	CRA6 (IRQA2)		CA_2 からの割込み要求なし，または CA_2 が出力に指定されている. (リセット状態)	ペリフェラル・コントロールライン CA_2 からの割込み要求あり.
7	$IRQA_1$	CRA7 ($IRQA_1$)		CA_1 からの割込み要求なし.	ペリフェラル・コントロールライン CA_1 からの割込み要求あり.

表7-4　コントロール・レジスタB(CRB)

CRB		ビット		0	1
0	CB_1制御	CRB0 (CB_1割込み出力マスクビット)		ペリフェラル・コントロールライン CB_1 による割込み信号によりCRB7($IRQB_1$)はセットされるが$\overline{IRQB}$はマスクされ出力されない("High"のまま).	ペリフェラル・コントロールライン CB_1 による割込み信号によりCRB7($IRQB_1$)がセットされ$\overline{IRQB}$出力が"Low"になる.
1	CB_1制御	CRB1(CB_1アクティブエッジビット)		CRB7($IRQB_1$)をCB_1入力の立ち下がりでセット.	CRB7($IRQB_1$)をCB_1入力の立ち上がりでセット.
2	DDRB	CRB2		データディレクション・レジスタBを指定.	ペリフェラル・インターフェースレジスタBを指定.
3	CB_2制御	CRB3 (CB_2割込み出力マスクビット)	CB_2入力指定のとき	ペリフェラル・コントロールライン CB_2 による割込み信号によりCRB6($IRQB_2$)はセットされるが$\overline{IRQB}$はマスクされ出力されない("High"のまま).	ペリフェラル・コントロールライン CB_1 による割込み信号によりCRB6($IRQB_2$)がセットされ$\overline{IRQB}$出力が"Low"になる.
4	CB_2制御	CRB4(CB_2アクティブエッジビット)	CB_2入力指定のとき	CRB6($IRQB_2$)をCB_2入力の立ち下がりでセット.	CRB6($IRQB_1$)をCB_2入力の立ち上がりでセット.
3	CB_2制御	CRB3 (ハンドシェイクモード指定ビット)	CB_2出力指定のとき	CRB4＝0のとき ペリフェラル・コントロールラインCB_2はMPUがペリフェラル・インターフェース・レジスタBにデータを書込んだ後,最初のEパルスの立ち上がりで"Low". ペリフェラル・コントロールライン CB_1 信号のアクティブエッジでCRB 7(IRQB₁ がセットされたとき"High"となる.	CRB4＝0のとき ペリフェラル・コントロールラインCB_2はMPUがペリフェラル・インターフェース・レジスタBにデータを書込んだ後,最初のEパルスの立ち上がりで"Low". 書込みEパルス後の2番目のEパルスの立ち上がりで"High"となる.
4	CB_2制御	CRB4	CB_2出力指定のとき	CRB3をハンドシェイクモード指定ビットとして有効にする.	CRB3の値をそのままCB_2に出力する.
5	CB_2制御	CRB5 (CB_2入出力プログラムビット)		ペリフェラル・コントロールライン CB_2 を入力に指定.	ペリフェラル・コントロールライン CB_2 を出力に指定.
6	$IRQB_2$	CRB6 ($IRQB_2$)		CB_2 からの割込み要求なし, また CB_2 が出力に指定されている. (リセット状態)	ペリフェラル・コントロールライン CB_2 から割込み要求あり.
7	$IRQB_1$	CRB7 ($IRQB_1$)		CB_1からの割込み要求なし.	ペリフェラル・コントロールライン CB_1 からの割込み要求あり.

表7-5　割り込み入力 CA_1 と CB_1 の機能

CRA1 (CRB1)	CRA0 (CRB0)	割込み要求入力 $CA_1(CB_1)$	割込みフラグ CRA7(CRB7)	割込み要求入力 $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	↓ アクティブ	$CA_1(CB_1)$の立ち下がりでセット	マスク，$\overline{IRQ}$は“High”のまま
0	1	↓ アクティブ	$CA_1(CB_1)$の立ち下がりでセット	インタラプト・フラグ・ビット CRA7(CRB7)が“1”になったとき“Low”
1	0	↑ アクティブ	$CA_1(CB_1)$の立ち上がりでセット	マスク，$\overline{IRQ}$は“High”のまま
1	1	↑ アクティブ	$CA_1(CB_1)$の立ち上がりでセット	インタラプト・フラグ・ビット CRA7(CRB7)が“1”になったとき“Low”

＊1. ↑ 信号の立ち上がりを示す(“Low” → “High”)
2. ↓ 信号の立ち下がりを示す(“High” → “Low”)
3. インタラプト・フラグビットCRA7は，MPUがペリフェラル・インターフェース・レジスタAの読み出し動作によってクリアされる．同様にCRB7はペリフェラル・インターフェース・レジスタBの読み出しでクリアされる．
4. CRA0(CRB0)が，“0”のとき割り込みが生じても$\overline{IRQA}$($\overline{IRQA}$)は出力されない．次にCRA0(CRB0)が“1”になるとその立ち上がりで$\overline{IRQA}$($\overline{IRQB}$)が出力される．

表7-6　割り込み入力 CA_2 と CB_2 の機能 (CRA5, CRB5＝0)

CRA5 (CRB5)	CRA4 (CRB4)	CRA3 (CRB3)	割込み要求入力 $CA_2(CB_2)$	割込みフラグ CRA6(CRB6)	割込み要求出力 $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	0	↓ アクティブ	$CA_2(CB_2)$の立ち下がりでセット	マスク，$\overline{IRQ}$は“High”のまま
0	0	1	↓ アクティブ	$CA_2(CB_2)$の立ち下がりでセット	インタラプト・フラグ・ビット CRA6(CRB6) が“1”になったとき“Low”
0	1	0	↑ アクティブ	$CA_2(CB_2)$の立ち上がりでセット	マスク，$\overline{IRQ}$は“High”のまま
0	1	1	↑ アクティブ	$CA_2(CB_2)$の立ち上がりでセット	インタラプト・フラグ・ビット CRA6(CRB6)が“1”になったとき“Low”

＊1. ↑ 信号の立ち上がりを示す(“Low” → “High”)
2. ↓ 信号の立ち下がりを示す(“High” → “Low”)
3. インタラプト・フラグビットCRA6は，ペリフェラル・インターフェース・レジスタAの読み出し動作によってクリアされる．同様にCRB6はペリフェラル・インターフェース・レジスタBの読み出しでクリアされる．
4. CRA3(CRB3)が，“0”のとき割り込みが生じても$\overline{IRQA}$($\overline{IRQB}$)は出力されない．次にCRA-3(CRB-3)が“1”になるとその立ち上がりで$\overline{IRQA}$($\overline{IRQB}$)が出力される．

RA7だけでなく，CRA6のフラグもクリアされますので注意しましょう(もちろんBポートのときは同時にCRB7，CRB6も)．

●CA_2，CB_2の機能設定

PIAのプログラミングでもっとも理解しにくいのがCA_2，CB_2の制御で

表7-7　CA_2，CB_2の制御出力としての機能

モード	CRA5	CRA4	CRA3	CA_2	
				ク　リ　ア	セ　ッ　ト
ハンドシェイク	1	0	0	MPUがペリフェラル・インターフェース・レジスタAを読出したときのEパルスの立ち下がりで“Low”となる．	CA_1信号のトリガエッジで“High”となる。
パルス出力	1	0	1	MPUがペリフェラル・インターフェース・レジスタAを読出したときのEパルスの立ち下がりで“Low”となる．	MPUの読出しの動作終了後次のEパルスの立ち下がりで“High”となる．
CRA3コピー	1	1	0	“Low”	
	1	1	1	“High”	

モード	CRB5	CRB4	CRB3	CB_2	
				ク　リ　ア	セ　ッ　ト
ハンドシェイク	1	0	0	MPUのペリフェラル・インターフェース・レジスタBの書込み動作後，最初のEパルスの立ち上がりで“Low”となる．	インタラプト・フラグ・ビットCRB7がCB_1のトリガエッジでセットされたとき“High”となる．
パルス出力	1	0	1	MPUのペリフェラル・インターフェース・レジスタBの書込み動作後，最初のEパルスの立ち上がりで“Low”となる．	書込みEパルス後の2番目のEパルスの立ち上がりで“High”となる．
CRB3コピー	1	1	0	“Low”	
	1	1	1	“High”	

す．CA_2，CB_2はともに入力あるいは出力として動作します．また動作モードも３種類あり，それぞれに設定できます．

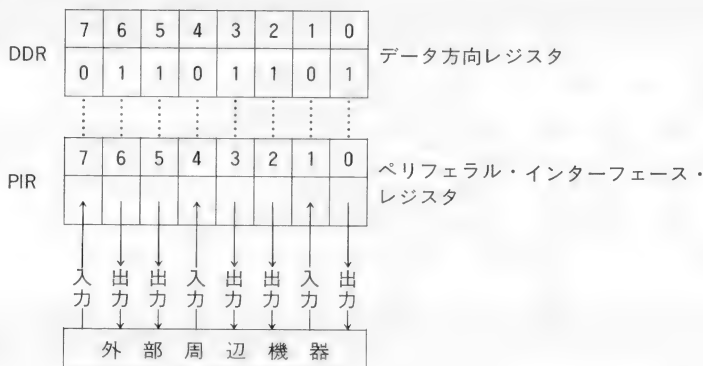
①ハンドシェイクモード

②パルス出力モード

③ＣＲＡ３（ＣＲＢ３）コピーモード

表7-7はモード別によるCA_2，CB_2の機能設定のためのビット設定値の一覧表です．

1・4　データ方向レジスタ

データ方向レジスタ(Data Direction Register＝ＤＤＲ)は，ペリフェラル・インターフェース・レジスタと周辺機器との間のデータのやりとりの方向を決定します．周辺機器とハード的に直接接続されているのはペリフェラル・インターフェース・レジスタ(ＰＩＲ)であり，データ方向レジスタではありません．

ＤＤＲとＰＩＲとの関係はＤＤＲ側が行動条件を示し，ＰＩＲ側がその行動条件に従い動作する形態になっています．ＤＤＲのあるビット位置の値が“０”のときはＰＩＲの同じビット位置にある端子は入力として，“１”のときは出力として動作します．**図7-5**にこの関係を示します．

このようにＰＩＲと周辺との間のデータのやりとりの方向を決定するのがＤＤＲの役割です．

1･5 PIRの役割

前項で述べましたが，このレジスタの先端はPA_0～PA_7またはPB_0～PB_7に接続してあり，さらにその先端は外部の周辺機器と接続しています．したがって，実質的にはこのＰＩＲが外部の周辺機器と接続しているといえます．

出力に指定したときはこのレジスタのｎｎビットめが“１”になると，ＰＡｎ（またはＰＢｎ）は“Ｈ”に，ｎビットめが“０”になるとＰＡｎ（ＰＢｎ）は“Ｌ”になります．ＰＩＲを入力に設定したときは周辺機器からのデータが入力されてＰＩＲに格納されます．

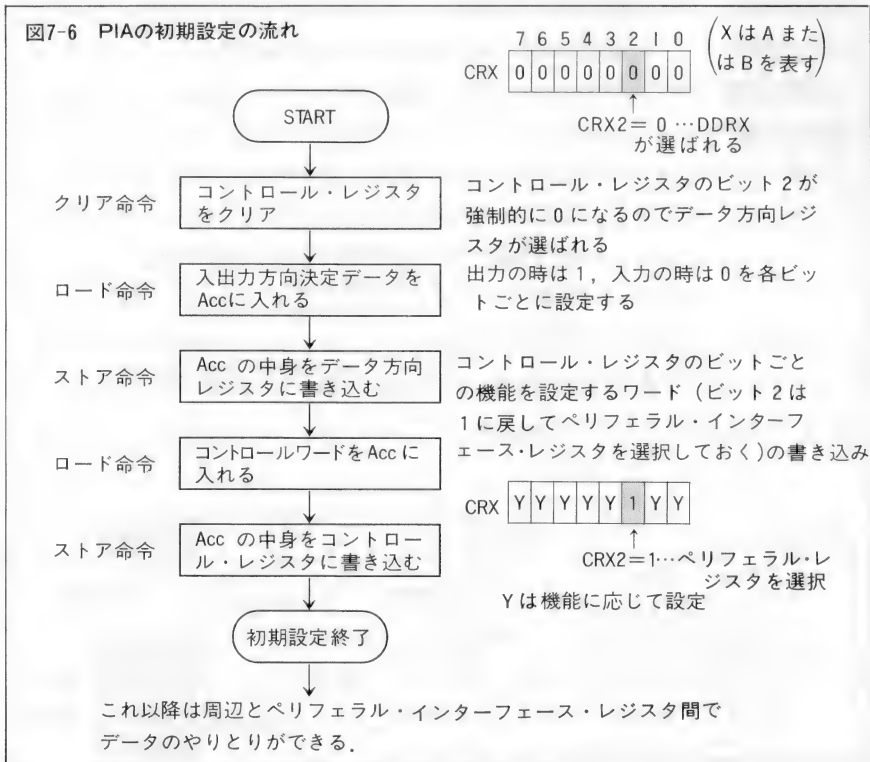
このレジスタを使用するときは前もってＤＤＲにより，データの入出力の方向を決定しておくようにします．

1・6 プログラミングの方法

●イニシャライズの場合

PIAのプログラムは必ずイニシャライズ（初期設定）から始まります．初期設定ルーチンでは，ペリフェラル・インターフェース・レジスタを入力用とするのか出力用とするのか，またはCA$_1$，CA$_2$，CB$_1$，CB$_2$の機能はどのようにセットするのか等の各種の設定をしておくことが必要です．初期設定ルーチンの流れを**図7-6**に示します．

データ方向レジスタとペリフェラル・インターフェース・レジスタの切り換えはコントロール・レジスタのビット2の値により行われます．したがって必ず最初はコントロール・レジスタに0を書き込み，強制的にビット2を0にしてデータ方向レジスタを選択します．次にデータ方向設定用データを書き込ん

図7-6　PIAの初期設定の流れ

でデータの入出力方向を決めたら，再びコントロール・レジスタのビット２を１にして，ペリフェラル・インターフェース・レジスタが選択されるようにしとおきます．

●ペリフェラル・コントロール・バス（CA_1，CA_2，CB_1，CB_2）を使用しないリード・ライト動作の場合

ＰＩＡを使用するときは大きく分けて２つの使用形態があります．

① PA_0～ＰＡｎ，PB_0～ＰＢｎの端子だけを使用して制御する

② ①の動作に加えてCA_1，CA_2，CB_1，CB_2などを使用する

まず①の使用形態の場合のプログラミング法を説明します．例としてＡポートの場合を取り上げますが，Ｂポートの場合も同様です．使用したレジスタ記号は次のようになります．

ＣＲＡ：コントロール・レジスタＡ　　ＣＲＢ：コントロール・レジスタＢ
ＤＤＲＡ：データ方向レジスタＡ　　ＤＤＲＢ：データ方向レジスタＢ
ＰＩＲＡ：ペリフェラル・インターフェース・レジスタＡ
ＰＩＲＢ：ペリフェラル・インターフェース・レジスタＢ

＜リード動作＞

Ａポートからの入力データを AccA に読み込む動作をプログラムする

```
CLR   CRA     *コントロール・レジスタのクリア
CLR   DDRA    *データ方向レジスタに０を書き込む
LDA   #$04    *ＰＩＲＡ選択に戻すためにコントロール
               ・レジスタＡに＄04を書き込む
STA   CRA
LDA   PIRA    *Ａポートに周辺装置からのデータが入力
               される
```

表7-8　コントロール・ワード＄04の意味

	$0				$4			
	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	0	0
ＣＲＡ	CA_1からのフラグ	CA_2からのフラグ	CA_2制御			DDRA選択	CA_1制御	
	CA_1の立ち下がりエッジで１になる	CA_2の立ち下がりエッジで１になる	CA_2を入力に	立ち下がりエッジでCRB7がセット	$\overline{\text{IRQA}}$は“H”	ペリフェラル・レジスタを選択	立ち下がりエッジでCRA7がセット	割り込み禁止

この動作設定用のコントロール・ワード＄04の意味は**表7-8**のようになります．

<ライト動作>

Aポートから＄55というデータを外部に出力する動作をプログラムしてみる．

```
CLR   CRA     *データ方向レジスタを選択した
LDA   #$FF
STA   DDRA    *Aポートは出力にセット
LDA   #$04    *PIRA選択に戻すためコントロール
              ・レジスタAに$04を書き込む
STA   CRA
LDA   #$55    *出力しようとするとデータを用意する
STA   PIRA    *Aポートから$55というデータが出力さ
              れる
```

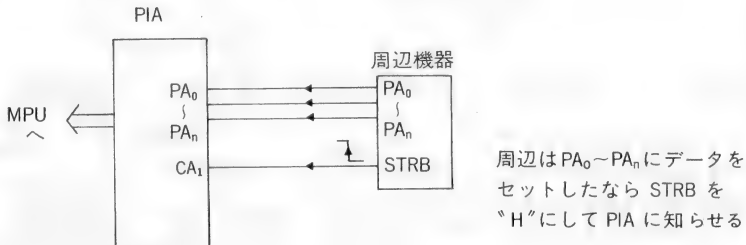
●ペリフェラル・コントロール・バスを使用するリード・ライト動作の場合

ペリフェラル・コントロール・バスの機能は非常に豊富です．というのは，ペリフェラル・コントロール・バスはタイミングによるデータのやりとりが本来の役目だからです．このことを理解するためには実際に制御対象を接続して，お互いの（PIAと制御対象間の）タイミングを計りながら行うことが必要です．

本項のスペースでは書き切れませんので，ここでは，ペリフェラル・コントロール・バスの基本的使用だけに限定して説明します．

<リード動作>

図7-7に結線図を示します．Aポートを使用して周辺からのデータをMPUにリードする．周辺機器からはCA_1に立ち上がり信号を与えてやり，データの発

図7-7　CA_1 を利用したデータ入力

生したことをＰＩＡに出力して知らせる．

```
         CLR   CRA     *DDRA選択
         CLR   DDRA    *Aポート入力に
         LDA   #$06    *コントロールワード設定
         STA   CRA     *CA1の立ち上がりでCRAAフラグを
                        セットする
L1       LDA   CRA     *CA1の入力があると，CRAの7ビット
                        めが1になるのでそれを待つ
         BPL   L1
         LDA   PIRA    *データがきたらAポートからデータを読む．
                        終わるとCRAフラグは“L”になる
```

この例におけるコントロールワード＄06の意味は**表7-9**のとおりです．このコントロールワードにおいては，コントロール・レジスタＣＲＡのビット０を１に変更すると，CA_1の立ち上がりですぐに割り込みルーチンに飛びます．しかし，本例では割り込みを禁止しています．

<ライト動作>………パルス出力モードによる例

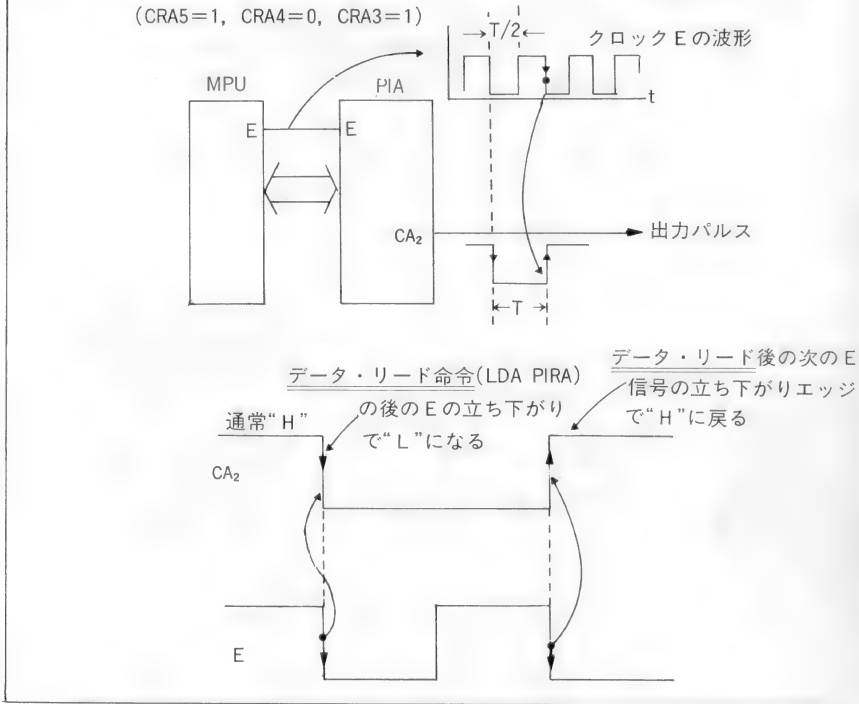
ＣＡ２からシステムクロックＥの周期の幅を持ったパルスを出力させるプログラムについて考えます．このような動作形態はパルス出力モードといいます．接続関係を**図7-8**に示します．

```
         CLR   CRA     *DDRAを選択
         CLR   DDRA    *Aポートは入力にセット
         LDA   #$2C    *コントロールワード・パルスモードにセ
                        ットする
         STA   CRA
         LDA   PIRA    *リードで“L”のパルスが出力，次のE
                        クロックの立ち下がりで“H”に戻る．
```

注意することは，このパルスモードのときはＡポート側とＢポート側は異な

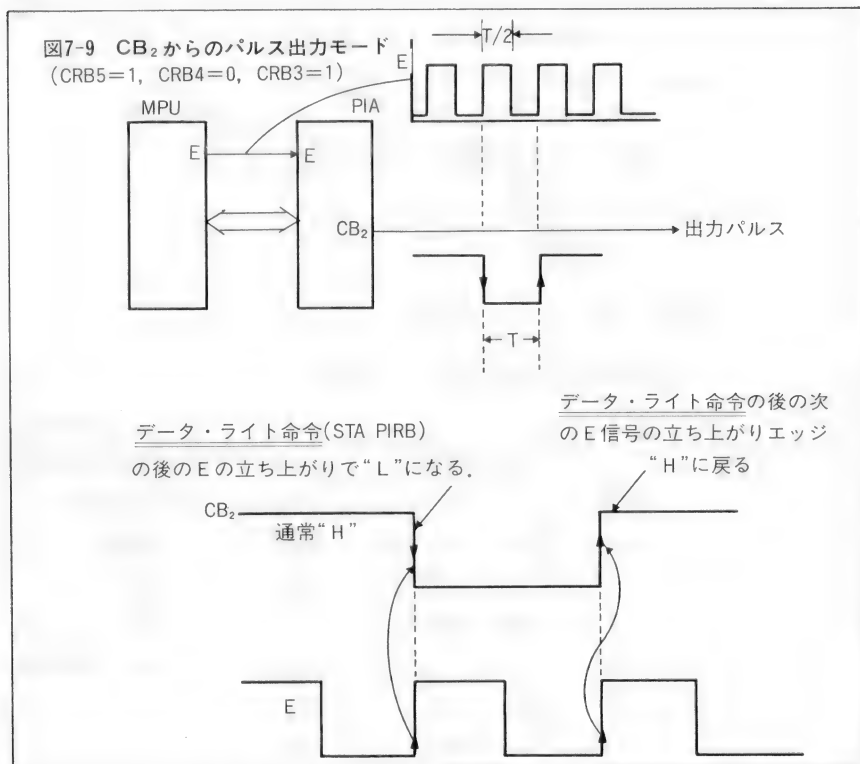
表7-9　コントロール・ワード＄06の意味

	$0				$6			
	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	1	0
CRA	CA_1からのフラグ	CA_2からのフラグ	CA_2制御			DDRA選択	CA_1制御	
	CA_7の立ち上がりでここが1になる	CA_2からの立ち下がりで1になる	CA_2を入力に	立ち下がりエッジでCRB 7がセット	$\overline{IRQA}$は“H”	ペリフェラル・レジを選択	立ち上がりでアクティブ	$\overline{IRQ}$はマスク

図7-8　CA_2からのパルス出力モード

った動作をすることです．Aポート側はリード命令を実行した後でパルスが出力されますが，Bポート側はライト命令を実行した後でパルスが出力されます．プログラムは次のようになります．

```
CLR    CRB     *DDRB選択
LDA    #$FF    *PIRBを出力にセット
STA    DDRB
LDA    #$2C    *コントロールワード・パルスモードにセット
STA    CRB
LDA    #$55    *ダミーのデータなのでこの場合なんでもよい
STA    PIRB    *CA2が"L"になり，この後のEクロッ
                クの立ち上がりで"H"に戻る
```

この場合のタイミング図を**図7-9**に示します．パルス出力モードでは出力パルス幅はクロックEの1周期幅しか出力できません．私もこのモードでプリンタ

図7-9　CB₂からのパルス出力モード
(CRB5=1, CRB4=0, CRB3=1)

のストローブ信号を出力するシステムを作りましたが，ストローブ信号のパルス幅を広くできないため，動作が不確実で困ったことがありました．

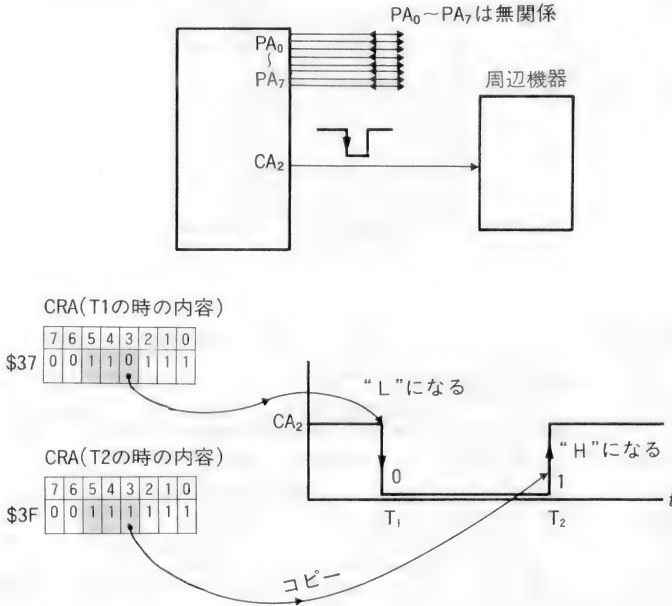
<ライト動作>………ＣＲＡ３コピーモードによる例

ＣＲＡ３コピーモードは私が勝手につけた名称ですが，動作の仕組みを一番よく表現しているものと自負しています．

このモードはCA_2からＣＲＡ３の値をコピーして出力します．ゆえにＣＡ２は１ビットの出力ポートとして使用できます．

このときPA_0～PA_7にはなんの影響もなくCA_2にのみ出力することができます．

```
CLR    CRA     *DDRAを選択
CLR    DDRA    *PA0～PA7を入力にセット
LDA    #$37    *CA2 "L" を出力
```

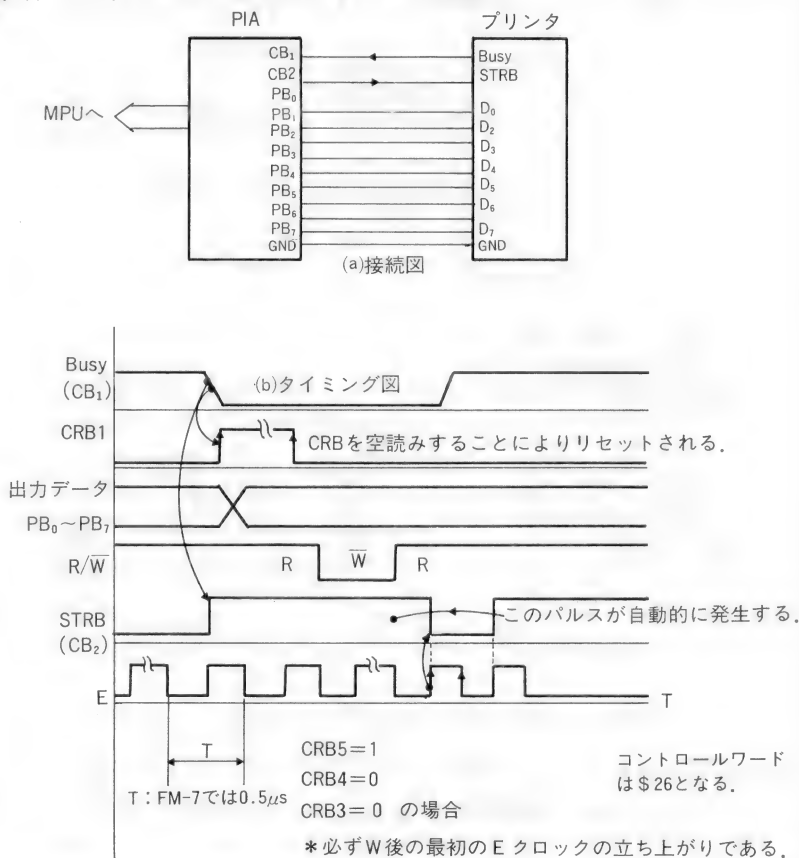
図7-10　CRA 3コピーモードの動作

```
STA   CRA
LDA   #$3F   *CA2から"H"を出力
STA   CRA
```

以上の動作の様子は**図7-10**のようになります.

<ライト動作>………ハンドシェイク・モードによる例

一般にMPUは高速動作ですが，入出力装置はメカニカルな部分が多くあるために非常に低速です．そこで，この両者がデータをやりとりするときはMPUが低速の入出力機器の動作を待つ場合が多くなりますが，速度の差が大きいので同期をとりながら動作することは大変です.

このようなときにMPU側と入出力側でお互いに“送信した”，“受信した”という信号（フラグ）を交わしながら，データのやりとりをすれば同期をとりながらの交信が可能になります．このようなモードをハンドシェイクモードと

図7-11　ハンドシェイクによるプリンタドライブの例

いいます.

ＰＩＡはこのハンドシェイクモードを自動的に行うオートマチックなハンドシェイク機能を備えています．したがってこのモードを使えばＭＰＵ側はいちいちデータの装備や送信終了などの確認をすることは必要なく，ＰＩＡがすべて自動的に処理してくれます．ＰＩＡの持っているもっとも優れた機能の１つです．です．ただ難をいえばこのモード動作が理解しにくいことです．実際の動作状態におけるデバッグも，タイミングがからんでくるためにわかりにくいのが実情です.

では，実際にプリンタをＰＩＡに接続して，ハンドシェイクモードでドライブしたときの例をあげて解説します．接続回路を**図7-11**に示します．

プリンタ側では印字作業が終了して，次の文字の受け付けが可能になればBusyフラグを“Ｌ”にしてＰＩＡに知らせます．ＰＩＡ側は，この信号によりＣＲＢ７のフラグをセットします．ＭＰＵはフラグを見ていてフラグがセットされると，ＭＰＵは印字するデータをＰＩＲＢに書き込みます．自動的にCB_2が“Ｈ”から“Ｌ”に変化して出力されます．

一方，Ａポートを用いてハンドシェイクを行うと(CB_1の代わりにCA_1，CB_2の代わりにCA_2）Ｂポートの場合とは異なった動作をするので注意が必要です．

Ｂポート側はペリフェラル・インターフェース・レジスタＢにデータを書き込むと，ＣＢ２が自動的に出力されましたが，Ａポート側はペリフェラル・インターフェース・レジスタＡを読み込むことによって，CA_2が出力されます．ハンドシェイク技法によりデータのやりとりをするときにはＡポートは入力用に，Ｂポートは出力用に設定するのが一般的です．次にＢポートを使用した場合の処理の中心となる部分のルーチンだけを示します．

```
L1    LDA    CRB     *CB1入力あったか（プリンタOKか）
      BPL    L1      *プリンタが準備されていなければ戻る
      LDA    PIRB    *CRB7をクリアするための空読み
      LDA    DATA    *印字データを AccA にセット
      LDA    PIRA    *印字データが出力され，同時にCB2から
                      ストローブ信号が自動的に出力される
```

このプログラムにおいて注意することがあります．ハンドシェイクモードにおいてはＣＲＢ７のリセットの動作がCB_2の出力動作よりも先行しなければならないことです．動作の順序が逆になりますとプログラムは動きません．このようなことが悩む原因となりやすいので，気を付けてください．

以上で簡単に68系の代表的入出力用ＬＳＩであるＰＩＡについて説明しました．非常に高い機能を持つＬＳＩであるため，使用法について書くだけでも一冊の本ができるといわれています．このＬＳＩのマスター法はたくさんの周辺機器を持続して実際に動作させた経験で体得することが一番です．読者のみなさんもＳＢＣ69に制御用機器を接続し，ぜひＰＩＡの動作法をマスターしてください．

ACIAのプログラミング

コントロール・レジスタ ステータス・レジスタ

ＡＣＩＡ（非同期通信インターフェース・アダプタ）は，パラレル↔シリアル・データ変換アダプタ用ＬＳＩです．最近脚光を浴びているデータ通信の分野では大活躍しており，これからも使用される製品が増えてくるでしょう．この機会にＡＣＩＡを使って，その動作原理を学んでください．

ここでは接続回路については６章で述べましたので，ここではＡＣＩＡの特性とプログラムの方法について述べることにします．

2・*1*　ACIAの素顔

ＡＣＩＡ(Asynchronous Communications Interface Adapter)は名前の示すとおり，単なるパラレル↔シリアル・データの変換アダプタ用ＬＳＩです．通信回線の制御まで行う高い機能を持っています．今回はＡＣＩＡとして68系ファミリのＬＳＩであるＭＣ6850を使用します．68系マイクロコンピュータのファミリＩＣにおいては，ＰＩＡとともに中心的な存在になっているＬＳＩです．シリアルデータ通信用のＬＳＩでは80系の8251Ａ，68系の6850, Z-80のSIO，この３つが代表的なものといえましょう．

6850はシステムクロック・スピードにより次の３種類があります．

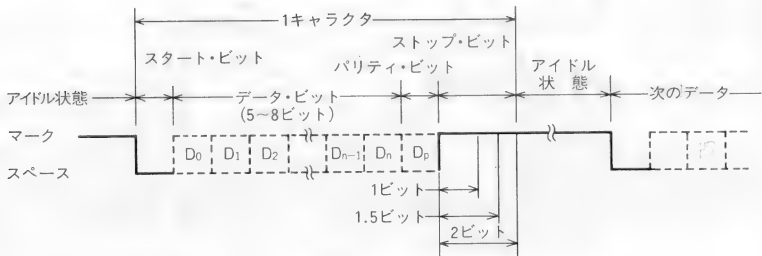
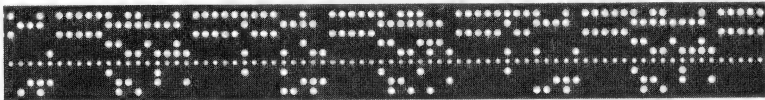
ＭＣ6850	1 MHz用
ＭＣ68Ａ50	1.5 MHz用
ＭＣ68Ｂ50	2 MHz用

本機では１ＭＨzバージョンのＭＣ6850を使います．

column
コ・ラ・ム

パリティ・ビット

エラー防止に用いる伝送誤まり制御の一方式です。送信側では送信するキャラクタ・データにパリティ・ビットという1ビットを付加して送出します．パリティには2種類あります．①奇数パリティ　②偶数パリティ．たとえば①の奇数パリティのときにはキャラクタを構成しているビット列にパリティ・ビットを付加したときの1の数が奇数になるようにパリティ・ビットの値を決めたものです．受信側ではパリティ・ビットを含めたキャラクタ内の1の数を数えて奇数になれば受信データは正しいと判断します．しかし，2つのビットが同時に反転した場合には正しいと判断されてしまうため，あまり検出能力が高いとはいえません．下図に非同期通信方式のデータ・フォーマットを示します．

また偶数パリティ形式データの例としてテープ・データを示します．紙テープでは穴が1つ1ビットに相当します．横1列のデータ中の穴の数がどこも偶数になっているのがわかるでしょう．この紙テープは7ビット・データでは偶数パリティ形式になっています．

7ビット・データ，偶数パリティ形式テープ・データ

#1

2•2　ACIAの構成

ACIAの内部構成は**図7-12**のようになります．ACIAのピン配置を**図7-13**に示します．図中の“直↔並変換器”はシフトレジスタのようなものです．入出力ポートにはラッチが入っていて，データの時間的効率が最大になるように動作します．このような形式はダブルバッファ方式と呼ばれ，MPUの利用効率が向上します．

2•3　コントロール・レジスタ

ACIAの内部レジスタは**表7-10**に示します．次にコントロール・レジスタについて説明します．

図7-12　ACIAの内部構成

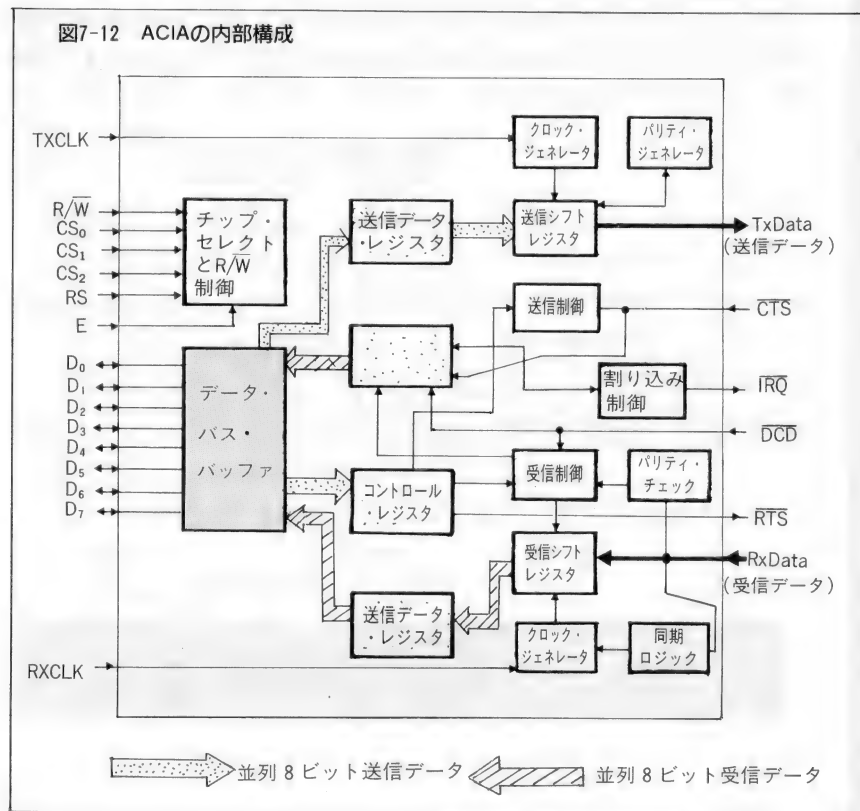
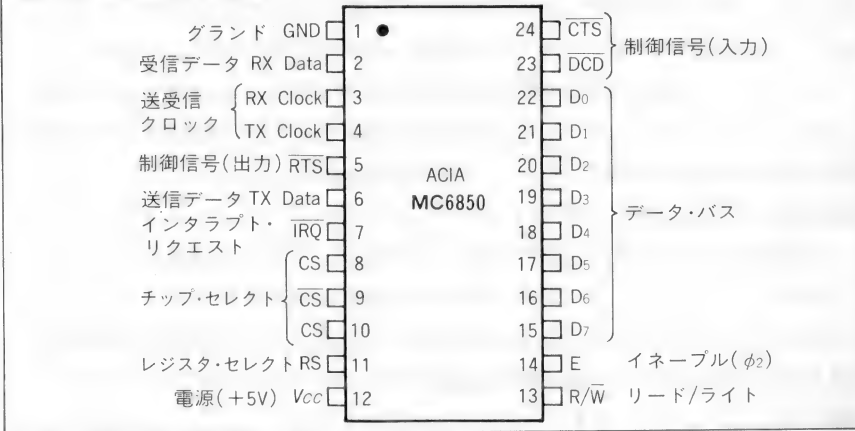



図7-13 ACIAのピン配置

コントロール・レジスタはACIAが送信あるいは受信するデータの型，割り込み制御の有無，クロックの分周比等についての取り決めを決定するところです．コントロール・レジスタのそれぞれのビットの意味は**表7-10**のようになります．

ではコントロール・レジスタのビットの役割を簡単に説明していきます．

●CR_1，CR_0（ビット1，0）

ACIAに与えられるクロックを，ACIAの伝送スピード（ボーレイト）に合う内部クロックに分周するビットです．

表7-10　コントロール・レジスタの各々のビットの意味

<table>
<tr><th rowspan="2">CR7</th><th rowspan="2">CR6</th><th rowspan="2">CR5</th><th colspan="2">ファンクション</th></tr>
<tr><th>$\overline{RTS}$</th><th>送信割込</th></tr>
<tr><td rowspan="3">1で受信時割込可</td><td>0</td><td>0</td><td>“L”</td><td>禁止</td></tr>
<tr><td>0</td><td>1</td><td>“L”</td><td>可</td></tr>
<tr><td>1</td><td>0</td><td>“H”</td><td>禁止</td></tr>
<tr><td rowspan="2">0で受信時割込禁止</td><td rowspan="2">1</td><td rowspan="2">1</td><td>“L”</td><td>禁止</td></tr>
<tr><td colspan="2">ブレークレベルを送信</td></tr>
</table>

<table>
<tr><th rowspan="2">CR4</th><th rowspan="2">CR3</th><th rowspan="2">CR2</th><th colspan="3">ファンクション</th></tr>
<tr><th>データビット</th><th>パリティ</th><th>ストップ・ビット</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td rowspan="4">7ビット</td><td>偶数</td><td rowspan="2">2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>奇数</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>偶数</td><td rowspan="2">1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>奇数</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td rowspan="4">8ビット</td><td rowspan="2"></td><td>2</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>偶数</td><td rowspan="2">1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>奇数</td></tr>
</table>

CR1	CR0	ファンクション
0	0	1/1
0	1	1/16
1	0	1/64
1	1	マスタリセット

たとえばクロック（Rxclock Txclock）が19200H_ZでCR_1，CR_0が0，1のとき，分周比は表を見ると1/16なので，内部では19200÷16＝1200になっています．この1200が**ボーレイト**であり，1200ボーのスピードといいます．

このビットが両方とも1のときはマスタ・リセットといい，ACIA内のレジスタをリセットします．ACIAはハード的なリセット端子がないので，使用するときは必ずマスタ・リセットを最初に行います．

●CR_4，CR_3，CR_2（ビット4，3，2）

送受信データの形式を決定するビット列です．ビット4が0でデータのビット数は7，ビット4が1でビット数は8の構成となります．ビット3，2はパリティ・ビットを付加するか，付加するとすればパリティは奇数か偶数かの決定とストップ・ビットの数の決定に関係します．

●CR_6，CR_5（ビット6，5）

送信時の割り込み機能とピン端子$\overline{\mathrm{RTS}}$の制御をするビットです．ビット5が0で送信時の割り込み禁止，1で送信時の割り込みがイネーブルとなります．MC6850には1ビット出力ポートとして$\overline{\mathrm{RTS}}$が，また1ビット入力ポートの$\overline{\mathrm{CTS}}$と$\overline{\mathrm{DCD}}$がそれぞれ用意されています．ビット6は出力ポートとして$\overline{\mathrm{RTS}}$端子を“L”，“H”のどちらかに設定します．ビット6が0で$\overline{\mathrm{RTS}}$は“L”，ビット6が1で“H”になります．

この機能により通信回線の制御ができます．ビット6，5の両方が1になると$\overline{\mathrm{RTS}}$は“L”になって送信時の割り込みは禁止になります．また“ブレーク・レベルのみの送信”という動作を行います．このときは送信データに関係なく，“L”のレベルの送信が続きます．

●CR_7（ビット7）

このビットが1になっていると，受信データの入力により割り込みが発生します．もしデータの入力処理を割り込みルーチンで実行させようとするならば，このビット7を1にしておきます．しかし，本機では割り込み処理をしていないので，ビット7は0にします．

以上，コントロール・レジスタの各ビットを説明しました．ユーザはプログラムのACIAのイニシャライズ時に自分のデータ伝送方式に合わせて各ビットをセットします．この操作は最初に1回のみ実行するだけです．

2・4　ステータス・レジスタ

ステータス・レジスタは送信部，受信部のいろいろなフラグが集まっているところです（表7-11）．各フラグは次のような働きをします．

●ＲＤＲＦ（Receive Data Register Full）

受信データ・レジスタ内に，受信したデータが存在することを示します．ＭＰＵが受信データ・レジスタを読み出すと，このフラグは自動的に０になります．つまり，１のときはデータがあることを意味します．

●ＴＤＲＥ（Transmit Data Register Empty）

送信データ・レジスタの中にデータがまだあるかないかを示します．１のときは送信データ・レジスタが空であることを意味し，ＭＰＵは次のデータをＡＣＩＡに送ってよいことになります．

●$\overline{\text{DCD}}$（Data Carrier Detect）

このフラグはＡＣＩＡの$\overline{\text{DCD}}$端子の状態をコピーしています．モデムからのキャリア入力がなくなると，この端子は“Ｈ”になります．“Ｌ”の状態で受信用のレジスタが動作可能となります．当然このときＲＤＲＦフラグは０になっています．ユーザが受信動作を実行しようとするときは$\overline{\text{DCD}}$端子の“Ｌ”を確認してから行います．$\overline{\text{DCD}}$フラグか１になっているとマスタ・リセットで“Ｌ”にすることはできません．

●$\overline{\text{CTS}}$（Clear To Send）

モデムからの$\overline{\text{CTS}}$入力をコピーします．$\overline{\text{CTS}}$端子が“Ｌ”のとき０にな

表7-11　ステータス・レジスタの各々のビットの意味

ファンクション／ビット位置	フラグ名	機　能	方　向
0	RDRF	受信データ・レジスタ内にデータあり	受
1	TDRE	送信データ・レジスタは空	送
2	$\overline{\text{DCD}}$	キャリア入力あり	受
3	$\overline{\text{CTS}}$	送信路の用意ができた	送
4	FE	フレーミング・エラー発生	受
5	OVRN	オーバーラン・エラー発生	受
6	PE	パリティ・エラー発生	受
7	$\overline{\text{IRQ}}$	割り込み要求あり	受・送

り，“H”になると1になります．$\overline{CTS}$が“H”のときTIREフラグが立ちませんので，送信動作ができません．送信動作を実行するときは必ず“L”にしておいてください．このフラグもマスタ・リセットされませんので注意が必要です．

●FE（Framing Error）

受信データにフレームエラーが生じたことを示します．

●OVRN（receiver OVerRuN）

受信データをMPUが読み込まないうちに次のデータが入力されてしまい，前のデータが失われたことを示します．CR_7＝1にしておくと，このときに割り込みの発生が生じます．この割り込みはマスタ・リセットまたはデータ・レジスタの読み込みによりクリアします．

●PE（Parity Error）

受信データにパリティ・エラーが発生したことを示します．

●$\overline{IRQ}$（Interrupt ReQuest）

このビットが1になっているときは送信，受信のどちらかで割り込み要求が発生したことを示します．フラグを調べることにより，割り込み発生の原因がTDRE，RDRF，$\overline{DCD}$，OVRNのどれによるのかが判断できます．

2・5　プログラミングの方法

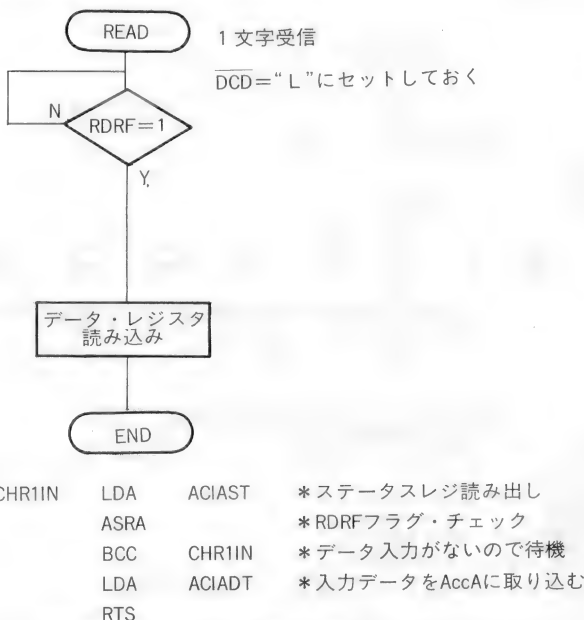
プログラミングはPIAのときと同様に必ずイニシャライズから始めます．前述しましたが，ACIAにはハード的なリセット端子はありませんので（電源投入直後はACIA内部の回路でリセットするが），ACIAの設定はマスタ・リセットを実行させてから行うのがよいでしょう．

① 最初にACIAのイニシャライズを実行してみましょう．

例として8ビット・データ，パリティなし，2ストップビット，1/16分周のモードに設定する場合は次のようになります．

```
LDA   #$03     *マスタ・リセット
STA   ACIACT   *コントロール・レジスタに書く
LDA   #$11     *コントロールワード値を
STA   ACIACT   *コントロール・レジスタに書く
```

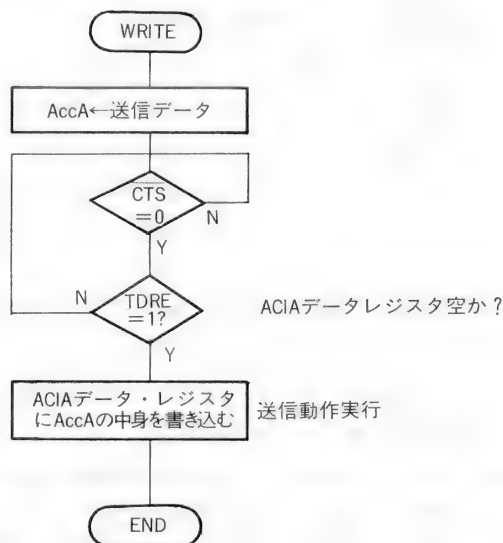
② 次に1文字受信の場合のフローチャートとプログラムを図7-14に示します．

図7-14 1文字受信ルーチンのフローチャート

ＡＣＩＡにシリアル・データが入ってくればＲＤＲＦフラグが１になるので，右に１ビットシフトします．ＲＤＲＦフラグはビット０ですので右へシフトさせるとキャリーが立つのでキャリーが１になるのを待ちます．キャリーが立ったらデータ入力があったことを意味しています．次にＡＣＩＡのデータ・レジスタから受信データを AccA にリードします．

③　１文字送信の場合のフローチャートとプログラムを**図7-15**に示します．送信手順はまず$\overline{\mathrm{CTS}}$の確認から始めます．$\overline{\mathrm{CTS}}$端子が“H”になっているとＴＤＲＥフラグが立たないので，必ず$\overline{\mathrm{CTS}}$端子は“L”にしておくようにします．

一般にはＲＴＳ端子を$\overline{\mathrm{CTS}}$に結線しておくので，ＡＣＩＡの初期設定のときはＲＴＳを“L”にしておけば自動的に$\overline{\mathrm{CTS}}$は“L”にすることができます．

プログラムとしては，まず$\overline{\mathrm{CTS}}$の確認をした後にＴＤＲＥフラグのチェックを行います．チェック方法はＡＳＲ命令を２回実行してＣフラグにビット１

図7-15　1文字送信ルーチンのフローチャート

```
        LDA    DATA         *送信データセット
SROUT   LDB    ACIAST       *ステータス読み込み
        BITB   #%00001000   *CTS=“L”を確認
        BNE    SROUT
        ASRB                *TDREフラグをキャリー
        ASRB                 ビットへ追い込む
        BCC    SROUT        *フラグ立たないなら待機
        STA    ACIADT       *送信データ出力
        RTS
```

の値を追い込んで，ＢＣＣ命令で判断します．このチェックをパスしたら前もってAccAにセットしておいた１バイトの送信用データをＡＣＩＡのデータ・レジスタにストアしてやると，ＡＣＩＡが自動的に送信用データを Tx Data の端子から外部に送信を行います．

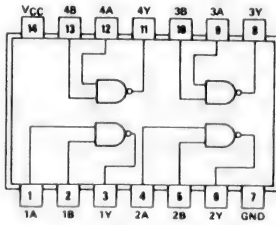
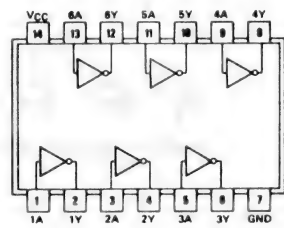
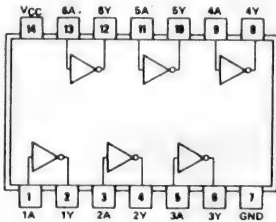
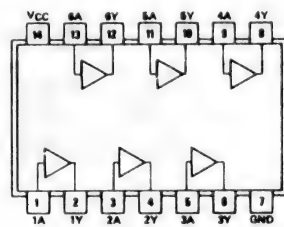
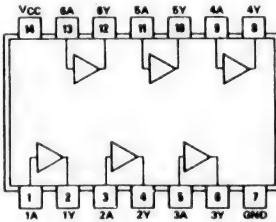
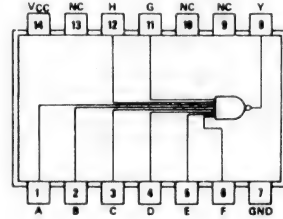
APPENDIX

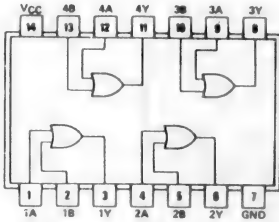
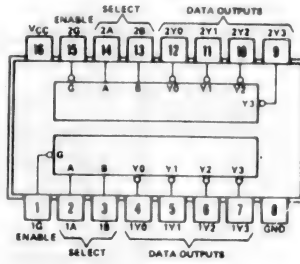
主要I/Oデバイス

6809命令セット

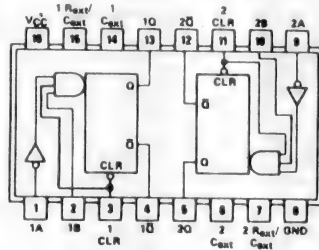
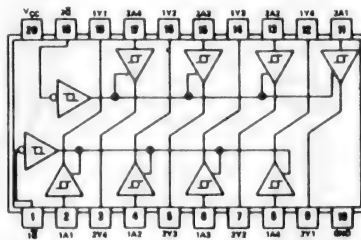
6809機能別命令表

1 ■主要デバイス

・使用したLS-TTLのピン接続

・使用したLS-TTLのピン接続

7432

74139

74123

FUNCTION TABLE

INPUTS			OUTPUTS	
CLEAR	A	B	Q	$\overline{Q}$
L	X	X	L	H
X	H	X	L*	H*
X	X	L	L*	H*
H	L	↑	⊓	⊔
H	↓	H	⊓	⊔
↑	L	H	⊓	⊔

74244

・使用したLS-TTLのピン接続

74245

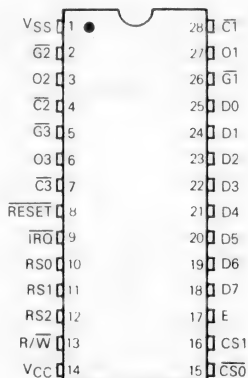
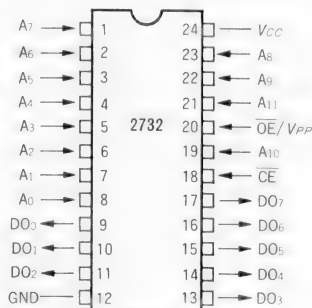
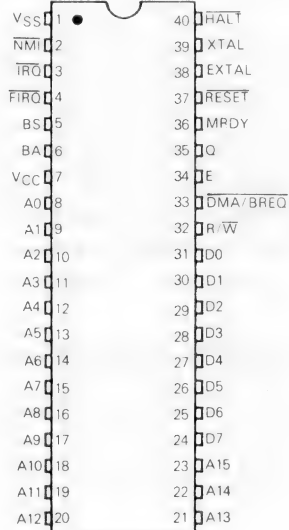
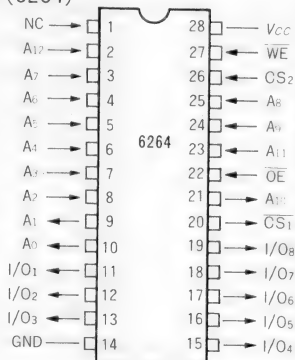
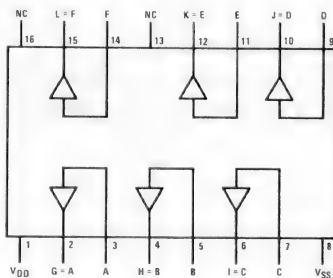
・PIAとACIAのピン配置

PIA (6821)

信号	ピン	ピン	信号
V_{SS}	1	40	CA1
PA0	2	39	CA2
PA1	3	38	$\overline{IRQA}$
PA2	4	37	$\overline{IRQB}$
PA3	5	36	RS0
PA4	6	35	RS1
PA5	7	34	$\overline{RESET}$
PA6	8	33	D0
PA7	9	32	D1
PB0	10	31	D2
PB1	11	30	D3
PB2	12	29	D4
PB3	13	28	D5
PB4	14	27	D6
PB5	15	26	D7
PB6	16	25	E
PB7	17	24	CS1
CB1	18	23	$\overline{CS2}$
CB2	19	22	CS0
V_{CC}	20	21	R/$\overline{W}$

ACIA (6850)

信号	ピン	ピン	信号
V_{SS}	1	24	$\overline{CTS}$
Rx Data	2	23	$\overline{DCD}$
Rx CLK	3	22	D0
Tx CLK	4	21	D1
$\overline{RTS}$	5	20	D2
Tx Data	6	19	D3
$\overline{IRQ}$	7	18	D4
CS0	8	17	D5
CS2	9	16	D6
CS1	10	15	D7
RS	11	14	E
V_{CC}	12	13	R/$\overline{W}$

・PTM，6809MPU，ROM，RAM，4050のピン配置

2 ■6809の命令表

・命令表の見方

マイクロコンピュータのプログラムを作成する場合には，１つ１つの命令の性格をよく理解しておかないと正確なプログラムを書くことはできません．個々の命令を理解するためには命令一覧表を参照することが必要となります．表❶を例として，命令表の見方，使い方を説明します．

プログラムを書くときに“LDA ＃$10”などと書きますが，このように書かれた記号をニーモニック・コードといいます．このニーモニック・コードをマシン語に変換するときは，まず命令表でニーモニック・コードが一致する行を探します．次にアドレッシングモードの一致する列の場所（OPコードの列です）に書かれている記号が目的の命令コードですので，これをマシン語として記入します．このときアドレッシングモードの欄で“＃”の欄の数字は命令の構成バイト数を示しています．先ほどのニーモニック・コードの欄ではOPが$86，＃が２になっていますので，命令長は２バイト構成であり，結局マシン語は$86，$10となります．

同じくアドレッシングモードの欄で“～”の記号はマシンサイクルを示しています．マシンサイクルは，その命令を実行するためにマシンサイクルで何サイクル必要であるかを示します．68系ではマシンサイクルはクロック周波数と同じです．ここでは命令は２マシンサイクルですので，システムクロックが１MHzのときは２μsで実行が終了することになります．

また“～”と“＃”の項で，＋のついているものはインデックスト・アドレッシングや PSH，PUL 命令であることを意味しています．インデックスト・アドレッシングの種類やレジスタの種類により，命令長が記されている数字以上のバイト数にすることを示しているので，表❷のように確認することが必要です．

次にフラグの変化の欄を説明します．フラグの変化の欄には“↕”，“・”の２つの記号があります．“↕”の記号はフラグがその命令の実行後変化することを表します．一方，“・”の記号は，その命令の実行結果がフラグに何の影響も与えないことを示します．とくに条件分岐したいときは分岐命令の直前の命令のフラグ変化をじゅうぶんに吟味する必要があります．

じゅうぶんな説明ではありませんでしたが，命令表の使い方がわかったことと思います．全命令表は次ページ以降に示します．

表❶

命令	具体例	アドレッシング・モード															説明	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス			エクステンディッド			インハレント説明				H	N	Z	V	C
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#						
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A ∧ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B ∧ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC ∧ IMM → CC					7
ASL	ASLA													48	2	1	A, B, M: C ← b7 … b0 ← 0	8	↕	↕	↕	↕
	ASLB													58	2	1		8	↕	↕	↕	↕
	ASL				08	6	2	68	6+	2+	78	7	3					8	↕	↕	↕	↕
ASR	ASRA													47	2	1	A, B, M: b7 … b0 → C	8	↕	↕	•	↕
	ASRB													57	2	1		8	↕	↕	•	↕
	ASR				07	6	2	67	6+	2+	77	7	3					8	↕	↕	•	↕
BIT	BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3				Bit Test A (M ∧ A)	•	↕	↕	0	•
	BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3				Bit Test B (M ∧ B)	•	↕	↕	0	•
CLR	CLRA													4F	2	1	0 → A	•	0	1	0	0
																		•	0	1	0	0
					0F	6	2	6F	6+	2+	7F							•	0	1	0	0
	COMB													53	2	1	$\overline{B}$ → B					
	COM										73	7	3				$\overline{M}$ → M	8	↕	↕	↕	↕
CWAI		3C	≥20	2													CC ∧ IMM → CC Wait for Interrupt					7
DAA														19	2	1	Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA													4A	2	1	A − 1 → A	•	↕	↕	↕	•
	DECB													5A	2	1	B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	6A	6+	2+	7A	7	3				M − 1 → M	•	↕	↕	↕	•
EOR	EORA	88	2	2	98	4	2	A8	4+	2+	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
	EORB	C8	2	2	D8	4	2	E8	4+	2+	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	8	2													R1 ↔ R2[2]	•	•	•	•	•
INC	INCA													4C	2	1	A + 1 → A	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	6C	6+	2+	7C	7	3				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	6E	3+	2+	7E	4	3				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	AD	7+	2+	BD	8	3				Jump to Subroutine	•	•	•	•	•
LD	LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				M → A	•	↕	↕	0	•
	LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				M → B	•	↕	↕	0	•
	LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				M M + 1 → D	•	↕	↕	0	•
	LDS	10 CE	4	4	10 DE	6	3	10 EE	6+	3+	10 FE	7	4				M M + 1 → S	•	↕	↕	0	•
	LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				M M + 1 → U	•	↕	↕	0	•
	LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				M M + 1 → X	•	↕	↕	0	•
	LDY	10 8E	4	4	10 9E	6	3	10 AE	6+	3+	10 BE	7	4				M M + 1 → Y	•	↕	↕	0	•
LEA	LEAS							32	4+	2+							EA[3] → S	•	•	•	•	•
	LEAU							33	4+	2+								•	•	•	•	•
								30	4+	2+								•	•	↕	•	•
								31	4+									•	•	↕	•	

表❷

命令	具体例	アドレッシング・モード															説明	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス			エクステンディッド			インハレント説明				H	N	Z	V	C
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#						
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D + M M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A ∧ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B ∧ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC ∧ IMM → CC					7

命令コード ↑　マシンサイクル数 ↑　命令構成バイト数 ↑　2以上のバイト数になることを示す ↑　フラグの変化状況を示す

・6809の全命令表

命令	具体例	アドレッシング・モード															説明	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス1			エクステンディッド			インハレント				H	N	Z	V	C
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#						
LSL	LSLA													48	2	1	A, B, M: C ← b7 … b0 ← 0	•	↕	↕	↕	↕
	LSLB													58	2	1		•	↕	↕	↕	↕
	LSL				08	6	2	68	6+	2+	78	7	3					•	↕	↕	↕	↕
LSR	LSRA													44	2	1	A, B, M: 0 → b7 … b0 → C	•	0	↕	•	↕
	LSRB													54	2	1		•	0	↕	•	↕
	LSR				04	6	2	64	6+	2+	74	7	3					•	0	↕	•	↕
MUL														3D	11	1	A × B → D (Unsigned)	•	•	↕	•	9
NEG	NEGA													40	2	1	$\overline{A}$ + 1 → A	8	↕	↕	↕	↕
	NEGB													50	2	1	$\overline{B}$ + 1 → B	8	↕	↕	↕	↕
	NEG				00	6	2	60	6+	2+	70	7	3				$\overline{M}$ + 1 → M	8	↕	↕	↕	↕
NOP														12	2	1	No Operation	•	•	•	•	•
OR	ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A V M → A	•	↕	↕	0	•
	ORB	CA	2	2	DA	4	2	EA	4+	2+	FA	5	3				B V M → B	•	↕	↕	0	•
	ORCC	1A	3	2													CC V IMM → CC				7	
PSH	PSHS	34	5+[4]	2													Push Registers on S Stack	•	•	•	•	•
	PSHU	36	5+[4]	2													Push Registers on U Stack	•	•	•	•	•
PUL	PULS	35	5+[4]	2													Pull Registers from S Stack	•	•	•	•	•
	PULU	37	5+[4]	2													Pull Registers from U Stack	•	•	•	•	•
ROL	ROLA													49	2	1	A, B, M: C ← b7 … b0 ← C	•	↕	↕	↕	↕
	ROLB													59	2	1		•	↕	↕	↕	↕
	ROL				09	6	2	69	6+	2+	79	7	3					•	↕	↕	↕	↕
ROR	RORA													46	2	1	A, B, M: C → b7 … b0 → C	•	↕	↕	•	↕
	RORB													56	2	1		•	↕	↕	•	↕
	ROR				06	6	2	66	6+	2+	76	7	3					•	↕	↕	•	↕
RTI														3B	6 15	1	Return From Interrupt					7
RTS														39	5	1	Return from Subroutine	•	•	•	•	•
SBC	SBCA	82	2	2	92	4	2	A2	4+	2+	B2	5	3				A − M − C → A	8	↕	↕	↕	↕
	SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B − M − C → B	8	↕	↕	↕	↕
SEX														1D	2	1	Sign Extend B into A	•	↕	↕	0	•
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A → M	•	↕	↕	0	•
	STB				D7	4	2	E7	4+	2+	F7	5	3				B → M	•	↕	↕	0	•
	STD				DD	5	2	ED	5+	2+	FD	6	3				D → M M + 1	•	↕	↕	0	•
	STS				10 DF	6	3	10 EF	6+	3+	10 FF	7	4				S → M M + 1	•	↕	↕	0	•
	STU				DF	5	2	EF	5+	2+	FF	6	3				U → M M + 1	•	↕	↕	0	•
	STX				9F	5	2	AF	5+	2+	BF	6	3				X → M M + 1	•	↕	↕	0	•
	STY				10 9F	6	3	10 AF	6+	3+	10 BF	7	4				Y → M M + 1	•	↕	↕	0	•
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A − M → A	8	↕	↕	↕	↕
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B − M → B	8	↕	↕	↕	↕
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D − M M + 1 → D	•	↕	↕	↕	↕
SWI	SWI[6]													3F	19	1	Software Interrupt 1	•	•	•	•	•
	SWI2[6]													10 3F	20	2	Software Interrupt 2	•	•	•	•	•
	SWI3[6]													11 3F	20	1	Software Interrupt 3	•	•	•	•	•
SYNC														13	≥4	1	Synchronize to Interrupt	•	•	•	•	•
TFR	R1, R2	1F	6	2													R1 → R2[2]	•	•	•	•	•
TST	TSTA													4D	2	1	Test A	•	↕	↕	0	•
	TSTB													5D	2	1	Test B	•	↕	↕	0	•
	TST				0D	6	2	6D	6+	2+	7D	7	3				Test M	•	↕	↕	0	•

注：

1．この欄には基本サイクルとバイト数を示してあります．具体的な数値は，インデックス・アドレス・モードの表，表２の数値を加えると得られます．
2．Ｒ１とＲ２は，任意の８ビット・レジスタまたは16ビット・レジスタのペアです．
　８ビット・レジスタはＡ，Ｂ，ＣＣ，ＤＰ
　16ビット・レジスタはＸ，Ｙ，Ｕ，Ｓ，Ｄ，ＰＣ
3．ＥＡは実効アドレスです．
4．ＰＳＨとＰＵＬ命令は，１バイトをプッシュまたはプルするのに５サイクル＋１サイクル要します．
5．5(6)の意味は：（ブランチ命令において）ブランチしない場合５サイクル，ブランチする場合６サイクルです．
6．ＳＷＩはＩとＦビットをセットします．SWI２とSWI３はＩとＦに影響を与えません．
7．コンディション・コードは命令実行結果によって直接セットされます．
8．ハーフ・キャリ・フラグの値は未定数です．
9．特殊ケースーｂ７がセットされているとキャリがセットされます．

命令	具体例	アドレッシング・モード															説明	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス			エクステンディッド			インハレント説明								
		Op	～	#	Op	～	#	Op	～	#	Op	～	#	Op	～	#		H	N	Z	V	C
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4 +	2 +	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4 +	2 +	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4 +	2 +	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4 +	2 +	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6 +	2 +	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4 +	2 +	B4	5	3				A Λ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4 +	2 +	F4	5	3				B Λ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC Λ IMM → CC					7
ASL	ASLA													48	2	1	A	8	↕	↕	↕	↕
	ASLB													58	2	1	B c ← b7 … b0 ← 0	8	↕	↕	↕	↕
	ASL				08	6	2	68	6 +	2 +	78	7	3				M	8	↕	↕	↕	↕
ASR	ASRA													47	2	1	A	8	↕	↕	•	↕
	ASRB													57	2	1	B b7 … b0 → c	8	↕	↕	•	↕
	ASR				07	6	2	67	6 +	2 +	77	7	3				M	8	↕	↕	•	↕
BIT	BITA	85	2	2	95	4	2	A5	4 +	2 +	B5	5	3				Bit Test A (M Λ A)	•	↕	↕	0	•
	BITB	C5	2	2	D5	4	2	E5	4 +	2 +	F5	5	3				Bit Test B (M Λ B)	•	↕	↕	0	•
CLR	CLRA													4F	2	1	0 → A	•	0	1	0	0
	CLRB													5F	2	1	0 → B	•	0	1	0	0
	CLR				0F	6	2	6F	6 +	2 +	7F	7	3				0 → M	•	0	1	0	0
CMP	CMPA	81	2	2	91	4	2	A1	4 +	2 +	B1	5	3				Compare M from A	8	↕	↕	↕	↕
	CMPB	C1	2	2	D1	4	2	E1	4 +	2 +	F1	5	3				Compare M from B	8	↕	↕	↕	↕
	CMPD	10	5	4	10	7	3	10	7 +	3 +	10	8	4				Compare M:M + 1 from D	•	↕	↕	↕	↕
		83			93			A3			B3											
	CMPS	11	5	4	11	7	3	11	7 +	3 +	11	8	4				Compare M:M + 1 from S	•	↕	↕	↕	↕
		8C			9C			AC			BC											
	CMPU	11	5	4	11	7	3	11	7 +	3 +	11	8	4				Compare M:M + 1 from U	•	↕	↕	↕	↕
		83			93			A3			B3											
	CMPX	8C	4	3	9C	6	2	AC	6 +	2 +	BC	7	3				Compare M:M + 1 from X	•	↕	↕	↕	↕
	CMPY	10	5	4	10	7	3	10	7 +	3 +	10	8	4				Compare M:M + 1 from Y	•	↕	↕	↕	↕
		8C			9C			AC			BC											
COM	COMA													43	2	1	$\overline{A}$ → A	•	↕	↕	0	1
	COMB													53	2	1	$\overline{B}$ → B	•	↕	↕	0	1
	COM				03	6	2	63	6 +	2 +	73	7	3				$\overline{M}$ → M	•	↕	↕	0	1
CWAI		3C	≥20	2													CC Λ IMM → CC Wait for Interrupt					7
DAA														19	2	1	Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA													4A	2	1	A − 1 → A	•	↕	↕	↕	•
	DECB													5A	2	1	B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	6A	6 +	2 +	7A	7	3				M − 1 → M	•	↕	↕	↕	•
EOR	EORA	88	2	2	98	4	2	A8	4 +	2 +	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
	EORB	C8	2	2	D8	4	2	E8	4 +	2 +	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	8	2													R1 ↔ R2[2]	•	•	•	•	•
INC	INCA													4C	2	1	A + 1 → A	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	6C	6 +	2 +	7C	7	3				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	6E	3 +	2 +	7E	4	3				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	AD	7 +	2 +	BD	8	3				Jump to Subroutine	•	•	•	•	•
LD	LDA	86	2	2	96	4	2	A6	4 +	2 +	B6	5	3				M → A	•	↕	↕	0	•
	LDB	C6	2	2	D6	4	2	E6	4 +	2 +	F6	5	3				M → B	•	↕	↕	0	•
	LDD	CC	3	3	DC	5	2	EC	5 +	2 +	FC	6	3				M:M + 1 → D	•	↕	↕	0	•
	LDS	10	4	4	10	6	3	10	6 +	3 +	10	7	4				M:M + 1 → S	•	↕	↕	0	•
		CE			DE			EE			FE											
	LDU	CE	3	3	DE	5	2	EE	5 +	2 +	FE	6	3				M:M + 1 → U	•	↕	↕	0	•
	LDX	8E	3	3	9E	5	2	AE	5 +	2 +	BE	6	3				M:M + 1 → X	•	↕	↕	0	•
	LDY	10	4	4	10	6	3	10	6 +	3 +	10	7	4				M:M + 1 → Y	•	↕	↕	0	•
		8E			9E			AE			BE											
LEA	LEAS							32	4 +	2 +							EA[3] → S	•	•	•	•	•
	LEAU							33	4 +	2 +							EA[3] → U	•	•	•	•	•
	LEAX							30	4 +	2 +							EA[3] → X	•	•	↕	•	•
	LEAY							31	4 +	2 +							EA[3] → Y	•	•	↕	•	•

凡例：

OP	オペコード（16進）	$\overline{M}$	Mの補数	↕	テストの結果の影響を受ける
～	MPUサイクル数	→	転送方向	•	変化なし
#	バイト数	H	ビット3からの補助キャリ	CC	コンディション・コード・レジスタ
+	算術加算	N	負（符号ビット）	:	コンカチネーション
−	算術減算	Z	ゼロ	V	論理和 OR
•	乗算	V	オーバフロー2の補数	Λ	論理積 AND
		C	ALUからのキャリ	⊻	排他的論理和 EXOR

命　令	具体例	アドレッシングモード リラティブ OP	 ~5	 #	説　明	5 H	3 N	2 Z	1 V	0 C
BCC	BCC LBCC	24 10 24	3 5(6)	2 4	Branch C=0 Long Branch C=0	• •	• •	• •	• •	• •
BCS	BCS LBCS	25 10 25	3 5(6)	2 4	Branch C=1 Long Branch C=1	• •	• •	• •	• •	• •
BEQ	BEQ LBEQ	27 10 27	3 5(6)	2 4	Branch Z=1 Long Branch Z=0	• •	• •	• •	• •	• •
BGE	BGE LBGE	2C 10 2C	3 5(6)	2 4	Branch≥Zero Long Branch≥Zero	• •	• •	• •	• •	• •
BGT	BGT LBGT	2E 10 2E	3 5(6)	2 4	Branch>Zero Long Branch>Zero	• •	• •	• •	• •	• •
BHI	BHI LBHI	22 10 22	3 5(6)	2 4	Branch Higher Long Branch Higher	• •	• •	• •	• •	• •
BHS	BHS LBHS	24 10 24	3 5(6)	2 4	Branch Higher or Same Long Branch Higher or Same	• •	• •	• •	• •	• •
BLE	BLE LBLE	2F 10 2F	3 5(6)	2 4	Branch≤Zero Long Branch≤Zero	• •	• •	• •	• •	• •
BLO	BLO LBLO	25 10 25	3 5(6)	2 4	Branch lower Long Branch Lower	• •	• •	• •	• •	• •

命　令	具体例	アドレッシングモード リラティブ OP	 ~5	 #	説　明	5 H	3 N	2 Z	1 V	0 C
BLS	BLS LBLS	23 10 23	3 5(6)	2 4	Branch Lower or Same Long Branch Lower or Same	• •	• •	• •	• •	• •
BLT	BLT LBLT	2D 10 2D	3 5(6)	2 4	Branch<Zero Long Branch<Zero	• •	• •	• •	• •	• •
BMI	BMI LBMI	2B 10 2B	3 5(6)	2 4	Branch Minus Long Branch Minus	• •	• •	• •	• •	• •
BNE	BNE LBNE	26 10 26	3 5(6)	2 4	Branch Z=0 Long Branch Z≠0	• •	• •	• •	• •	• •
BPL	BPL LBPL	2A 10 2A	3 5(6)	2 4	Branch Plus Long Branch Plus	• •	• •	• •	• •	• •
BRA	BRA LBRA	20 16	3 5	2 3	Branch Always Long Branch Always	• •	• •	• •	• •	• •
BRN	BRN LBRN	21 10 21	3 5	2 4	Branch Never Long Branch Never	• •	• •	• •	• •	• •
BSR	BSR LBSR	8D 17	7 9	2 3	Branch to Subroutine Long Branch to Subroutine	• •	• •	• •	• •	• •
BVC	BVC LBVC	28 10 28	3 5(6)	2 4	Branch V=0 Long Branch V=0	• •	• •	• •	• •	• •
BVS	BVS LBVS	29 10 29	3 5(6)	2 4	Branch V=1 Long Branch V=1	• •	• •	• •	• •	• •

単純ブランチ

	OP	~	#
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LBRN	1021	5	4
BSR	8D	7	2
LBSR	17	9	3

単純コンディショナル・ブランチ（注１～４）

Test	True	OP	False	OP
N=1	BMI	2B	BPL	2A
Z=1	BEQ	27	BNE	26
V=1	BVS	29	BVC	28
C=1	BCS	25	BCC	24

符号化コンディショナル・ブランチ（注１～４）

Test	True	OP	False	OP
r>m	BGT	2E	BLE	2F
r≥m	BGE	2C	BLT	2D
r=m	BEQ	27	BNE	26
r≤m	BLE	2F	BGT	2E
r<m	BLT	2D	BGE	2C

符号なしコンディショナル・ブランチ（注１～４）

Test	True	OP	False	OP
r>m	BHI	22	BLS	23
r≥m	BHS	24	BLO	25
r=m	BEQ	27	BNE	26
r≤m	BLS	23	BHI	22
r<m	BLO	25	BHS	24

注：

1．すべてのコンディショナル・ブランチには，ショートとロングがあります．
2．すべてのショート・ブランチは2バイトであり，3サイクル要します．
3．すべてのコンディショナル・ロング・ブランチには，ショート・ブランチのオペコードに＄10がつき，ブランチ先を表すため16ビットのオフセットを使います．
4．すべてのコンディショナル・ロング・ブランチは4バイトで，ブランチする場合6サイクル，ブランチしない場合5サイクルを要します．

3 ■機能別命令表

・8ビット・アキュムレータ/メモリ命令

命　　令	説　　明
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A accumulator
DEC,DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A × B → D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

注：A, B, CC, それにDPはPSHS, PSHU（PULS, PULU）命令を用いてスタックにプッシュ（プル）できます.

・16ビット・アキュムレータ/メモリ命令

命　令	説　明
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U, or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U, or PC
TFR R, D	Transfer X, Y, S, U, or PC to D

注：DはPSHS，PSHU（PULS，PULU）命令を用いてスタックにプッシュ（プル）できます．

・インデックス・レジスタ/スタック・ポインタ命令

命　令	説　明
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, S, or PC onto user stack
PULS	Pull A, B, CC, DP, D, X, Y, U, or PC from hardware stack
PULU	Pull A, B, CC, DP, D, X, Y, S, or PC from hardware stack
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U, or PC
ABX	Add B accumulator to X (unsigned)

・ブランチ命令

命　　令	説　　明
	単純ブランチ
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
	符号付きブランチ
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid 2s complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid 2s complement result
BLT, LBLT	Branch if less than (signed)
	符号なしブランチ
BHI, LBHI	Branch if higher (unsigned)
BCC,LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
	その他ブランチ
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

・その他の命令

命　　　令	説　　　明
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

【引用・参考文献】
M6800ファミリ8ビット・データブック，日本モトローラ，1980
MC6809-MC6809Eマイクロプロセッサプログラミングマニュアル，日本モトローラ，1982
8-BIT MICROPROCESSOR & PERIPHERAL DATA, MOTOROLA，1983
日立データブック8/16ビットマイクロプロセッサ，日立，1981
日立データブック8/16ビットマイクロコンピュータ周辺LSI，日立，1981
The Bipolar Digital Integrated Circuits Data Book, TEXAS INSTRUMENTS, 1986
LOGIC DATABOOK VOLUME1, NATIONAL SEMICONDUCTOR CORPORATION, 1984
デバイスマニュアル，マキシム,・ジャパン
横井与次郎，マイクロコンピュータ・ハードウェア基礎技術，ラジオ技術社
横井与次郎，マイクロコンピュータ基礎技術マニュアル，ラジオ技術社
松本吉彦，私だけのマイコン設計&製作，CQ出版社
A，オズボーン，マイクロコンピュータプログラミング，マイテック社
相原隆文，Z-80実用マイコン製作，技術評講社
高橋豊，6809ソフトウェア開発，CQ出版社

おわりに

以上，MC6809を搭載したマイクロコンピュータについて，基礎理論と製作回路の両面から説明してきました．6809のイメージが湧いてきたでしょうか．

この本は上下巻合わせて効果を発揮します．第1巻は基礎編としての性格を持ち，第2巻は応用編としての性格を持っています．第1巻で学んだ内容をベースにして，いよいよ第2巻では実際にワンボード・マイコンを組み立てていきます．

本書は，“はじめに”でも述べたように，
「マイコンを理解する一番の近道は作ってみることである」
ということを理念に構成されています．ぜひ，第2巻のマイコン製作まで行うことを望みます．

わたしの経験でも，理論面でどうしても理解できなくて困ったことがありましたが，実際にコンピュータを作り，プログラムを実行していく段階で，ひとりでに疑問点が氷解してしまったものです．第1巻の内容がどうもよく理解できないという人も，第2巻に進むことで，さらに大きく前進できるのではないかと思います．

部品の発注は終わりましたか．まだ部品を注文していない人は，ぜひ部品をそろえるようにしてください．では，みなさん，第2巻でまたお会いしましょう．

さくいん

●数字順

●アルファベット順

●50音順

近藤元一（こんどう　もとかず）
山形県立東根工業高等学校　電子科教諭

HARDWARE BOOKS 3
6809マイコン製作実習（上）

昭和62年2月20日　初版 第1刷発行

著　者　近藤元一
発行者　片岡　巌
発行所　株式会社技術評論社
東京都千代田区九段南2-4-13
電話　03(262)9351　営業部
03(262)7671　編集部

印刷／製本　図書印刷

定価はカバーに表示してあります

ISBN4-87408-867-8 C3055

HARDWARE BOOKS 3

6809マイコン製作実習(上)

PRACTICE MAKING OF 6809 MICRO COMPUTER SYSTEM

ISBN4-87408-867-8 C3055 ¥1800E 定価1800円